吉里吉里/KAGではじめるゲーム制作

まえがき

「吉里吉里/KAG」に興味をもっていただき、ありがとうございます。

「吉里吉里」は、「TJS」というプログラミング言語を使ってマルチメディア タイトルを作るためのツールです。「KAG」はその「吉里吉里」をノベル・アドベンチャーゲームとして動作させるためのスクリプト群です。

この本は「吉里吉里/KAG」でゲームを作ろうと考えている方を対象にしています。前半、「KAG」についてPIA少尉さんに解説いただき、後半は私（W.Dee）が「TJS」について解説しています。

「KAG」は、ノベル・アドベンチャーゲームを、簡単なテキスト・ファイルの記述で制作できるのです。一方、「TJS」は、JavaやJavaScriptといった言語に似たプログラミング言語で、「KAG」よりもより「吉里吉里」本体やコンピュータ側に近い操作を行ないます。

「KAG」でできることは広く、その機能を知るにつれて、きっと想像力をかきたてられると思います。そのうえ、何か特別なことをやりたいと思った場合は、「TJS」を使ってより詳細に「吉里吉里」を操作することができます。また、KAG自体が「TJS」で記述されているため、改造が比較的容易です。「吉里吉里/KAG」においては、この「KAG」と「TJS」を組み合わせて使えることが、大きな特徴です。

できることが広いというと、何か難しそうで踏み入るには勇気が要るように思われるかもしれません。しかし、最初の一歩を踏み出せば、あとはどんどんと進んでいけるものだと思います。この本がその道しるべとなり、皆様のお役に立てることを願います。

謝辞

「吉里吉里」は、もともとは自分（W.Dee）自身が使うために作られ、「目的」を達成するための「手段」でした。それがいつのまにか「手段」から「目的」に変わり、ついにはこのたびのように本を書かせていただけることにまでなったのは、自分自身でも驚くべきことであると思っています。KAG編の執筆に快く応じてくださったPIA少尉さん、書籍への収録を許可してくださった方々に感謝いたします。

また、「吉里吉里/KAG」は私一人の力で作り上げられたものではありません。多くの方々のご意見をうけて作り上げられたものです。作品を世に公開された方々、作品を製作中の方々、デバッグに協力してくださった方々、ありがたい助言を下さった方々、奇行に付き合って下さった方々、「吉里吉里/KAG推進委員会」の方々、IRCチャンネル「#kirikiriirc」の方々、そしてこの本を読んでくださる方々に、深く感謝の意を表わしたいと思います。

2003年5月吉日 W.Dee

全国のクリエイターの皆さん、最近ゲームを作っていますか？

PIA少尉こと私は2000年の冬に『1999ChristmasEve』というテキストアドベンチャー・ゲームを公開しました。

ゲーム制作は初めてという素人集団を、ディレクションは初めてという少尉が指揮して完成させた稚拙な処女作なのですが、ベクターデザインさんのサイトにアップした結果、2年間で150,000ダウンロードを突破するという望外な反響をいただきました。

実はこのゲーム、「吉里吉里／KAG」と呼ばれるフリーソフトのアドベンチャー・ゲーム作成ツールで制作させてもらったのですが、ある日、当の「吉里吉里／KAG開発者」のW.Deeさんから一通のメールが舞い込んできました。

W.Dee	**：**	**少尉は、僕よりも「KAG」のことを知ってるでしょ**
少　尉	**：**	**は、はあ（そんなわけないと思うけどなぁ？）**
W.Dee	**：**	**『1999ChristmasEve』もとりあえず好評みたいだしね。**
少　尉	**：**	**お、おかげさまで。**
W.Dee	**：**	**というわけでお願いなんですけど、「吉里吉里／KAG」の使い方を全国のクリエイターさん向けの原稿として書いてくれません？**
少　尉	**：**	**ぎょえ！＼(><)／**

少尉の恩人でもあり師匠でもあるW.Deeさんの恩を裏切るわけにはまいりません。返事は一つでした。

こうして少尉は、自分にそんな大役が務まるのかも半信半疑のまま、原稿執筆を承諾したのであります。

少尉もまだまだKAG使いとしては半人前ですが、少しでも「本物のKAG使い」を目指して日夜KAGのシナリオ・スクリプトと取り組んでいる皆さんのお役に立てれば幸いです。

華麗なる21世紀のフルカラー対応AVG&ノベル作成ソフトを目指して躍進を続ける「吉里吉里／KAG」を、今後ともどうぞよろしくお願いします。

横浜かまいたちファンクラブ代表　PIA少尉　筆

吉里吉里/KAGではじめるゲーム制作

Step 5 選択肢とラベル

Step 6 変数の利用

Step 7 マクロを活用する

Step 8 クリッカブル・マップの利用

Step 9 その他の機能

Step 10 リリースに向けて

TJS編 W.Dee

8章 スコープ

9章 While文

10章 for文

11章 switch文

12章 オブジェクト

13章 配列

14章 クラス

15章 KAGの構造と動作

16章 KAG用プラグイン

付 録

Step 0

KAGを使う前に

この章では「吉里吉里/KAG」の紹介と、使う前に必要な最低限の準備について解説します。

0-1 「吉里吉里／KAG」とは？

「吉里吉里」は、「TJS」と呼ばれる独自のスクリプト言語を持つプログラム・エンジンです。

「吉里吉里」はエンジンなので、それ単体では何もしませんが、動作命令をプログラムした「TJSスクリプト」を与えてやることにより、Windows上で動作するさまざまなアプリケーションやゲームとして利用することができます。

ただ、TJS自体はJavaScriptに似ているとは言え、私のようなプログラマーでない一般庶民にはやや難しく感じられます。

「プログラムを組むプログラマー」—これは、時代がいくら変わっても、モノを創ることを夢見るクリエイターの卵にとっては、憧れの対象であることに変わりありません。

もちろん、プログラムがスイスイと書ければこんなに嬉しいことはありません。「あんな効果」や「こんな演出」なども、「よーし。おいらがチョチョイのチョイとプログラム書いて…」って、あなた、それが簡単にできれば誰も苦労はしません。

たしかに、「C++」*や「アセンブラ」*などのプログラム言語を駆使すれば何でもできるでしょうが、言語の特性や命令・構文などを熟知しなくては、最初の一歩すら踏み出せません。また、「別にプログラマーになるつもりはない。楽をしてカッコいいゲームを作りたいだけだ」という人には、この壁はチョモランマのように高く聳え立つ人跡未踏の（しかも苦労してまで登ろうとは思わない）冬山にすぎません。

＊C++
Bjarne Stroustrup氏が開発したC言語を改良してオブジェクト指向の機能を追加した言語。

＊アセンブラ
CPUが直接理解できる機械語のひとつひとつにニーモニック（略語）を割り当てたアセンブリ言語で書かれたプログラムを機械語に翻訳するもの。

そこで、W.Dee氏は、TJSを使い、ゲーム制作に必要と思われる命令群を持つキットを作りました。これが「KAG」です。

ちなみに、KAGは「Kirikiri_Adventure_Game_system」の略であり、正式には『KAGシステム』と呼びますが、ここでは便宜上「KAG」と呼ぶことにします。

KAGで書かれたスクリプトはKAGで解釈された後に、「TJS」に引き渡され、「吉里吉里」をアドベンチャー・ゲームやノベルゲームとして動作させることができます。

大切なことを言い忘れていましたが、「吉里吉里／KAG」はフリーソフトです。
フリーソフトはもちろん、市販品や同人作品など、お金を取るための作品に「吉里吉里／KAG」を利用しても、W.Deeさんにお金を払う必要はありません。

さあ、あなたもこの無料にして最高のゲーム制作ツール『吉里吉里／KAG』を使って、あなたの胸に秘めた想いを形にしてみませんか？

■「吉里吉里／KAG」の自由度の高さ

さて、私が吉里吉里／KAGのどこに心底惚れ込んでいるかといえば、ひとえに「自由度と難易度のバランスが絶妙」な点です。

普通、簡単なゲーム制作ツールは、簡単な故に制限が多いです。
たとえば、こんな感じですね。

・同時に発音できる効果音は2つまで
・変数は全部で255個まで
・画像を表示できる最大数は3枚まで

このように、“簡単操作”を謳う制作ツールに背後霊のようにくっついている「開発者の一方的都合による変な仕様」に泣かされたクリエイターも多いと思います。

もちろん、この背後霊的制限は、操作性や制作工程を簡易化するためには非常に大切なことです。でも、でもでもでもですよ、あまりにも制限ばかり目に付いてしまい、「あれもダメ」「これもダメ！」って言われたら、親の束縛から逃避するために家出を敢行する中学生ではありませんが、「やってられっかよ！」って制作する意欲も失せちゃいますよね。

私は「凝ったゲームを作りたい。でも、プログラムを勉強するほどじゃない。だからといって、制限の多い簡易ツールは使いたくないなぁ」という一人でした。ですから、吉里吉里／KAGに出会ったときには、心底驚きました。
ツールがゲーム制作者に課す“制限”と呼ばれるものがほとんど存在しなかったのですから。

「画像レイヤーを10枚使いたい？　どうぞ」
「変数を100個使いたい？　使う場所があれば、けっこうですよ」
「選択肢の数を自由に増やしたり減らしたりしたい？　問題ありませんわ」
「セーブ・データの保存可能数を50個にしたい？　まあ、ユーザー思いなことですね。もちろんかまいませんよ」
「メニューをいろいろと変更したい？　分かりました。システム的な部分も開放していますから、どうぞご自由に調整してください」

まるで不可能という言葉が存在しない辞書をもつ、明眸皓歯で才色兼備な完全無欠の美貌秘書（しかも物分かりが良い）のようではありませんか。
こんな相手に惚れないわけにはいきませんでした。

> **コラム～少尿のひとりごと**
> KAGに惚れた惚れたと言いつつも、私には8つ年下の妻がおりまして、実は自称愛妻家なのです。

■「吉里吉里／KAG」の敷居の低さと奥の深さ

皆さんは、「制限が存在しないってことは、相当難解なスクリプトを学ばないといけないんだろうなぁ」と思っているでしょう。
さにあらず。「吉里吉里／KAG」は、初心者であっても取り扱いは簡単です。

一判断として、「FrontPageExpress」*や「HomePageBuilder」*のようなアプリケーションソフトを使わずにhtmlを手書きできる人なら、KAGを使ってすぐにゲームが作れます。なぜ断定的な物言いができるかといえば、私自身がそうだったからです。

* FrontPageExpress MicrosoftCorporationのホームページ作成ソフト。

* HomePageBuilder IBMのホームページ作成ソフト。

もちろん、「吉里吉里／KAG」を用いてゲームを制作する場合、「簡単作成ツール」などと銘打たれているソフトを使うより勉強量は増えます。でも、万能無敵な演出効果と美しい画像処理、そして開発者が涙するほどのきめ細やかなチューニング＆カスタマイズ機能を目の当たりにすれば、あなたは絶対に「吉里吉里／KAG」を手放したくなくなるでしょう。

私自身、最初は基本的なコマンド・タグ（命令）のみを使ってゲームの骨組みを構築していました。初めて作ったものは、画面切り替えも変数処理もない、ただ読ませるだけの絵本でした。

ですが、開発を進めていくにつれて演出に対する希望や要求が増えてきたので、その場その場で実現したい演出に必要なコマンドを少しずつ勉強してきたのです。そして「吉里吉里／KAG」は、見事にその要求に応えてくれました。

つまり、最初にすべてを覚えるのではなく、自分に必要なコマンドを自分のペースで段階的に学んでいけるのが、「吉里吉里／KAG」の意外なウリでもあり、開発者に対する優しさなのです。

このように、「吉里吉里／KAG」は、プログラムの書けないクリエイターにはうってつけのバランスの取れたスクリプト・ツールですが、バリバリとプログラムを書いている人にも朗報があります。

「吉里吉里」の上で動作する「KAG」は、前述した通り吉里吉里の専用言語「TJS」で書かれているため、動作が気に入らなかったり、提供されていない機能を追加したい場合には、KAG自体を「TJS」レベルでカスタマイズすることが可能です。

もちろん、わざわざ「KAG」をカスタマイズしなくても、オリジナルの「TJS式」を「KAGスクリプト」内に書くことでTJSを使った演出は可能になります。背景画像の上に雪や雨など（好みによっては槍なども）を降らせたり、クリッカブル・マップを使ったり、複雑な画面処理を行なったり…と、市販ゲームに実装されているたいていの処理は可能になります。

コラム～少尉のひとりごと

KAGのカスタマイズについてですが、「TJSが理解できる人はKAG自体も好きなようにカスタマイズしてください。どんどんしちゃってください」と開発者のW.Deeさんは申しております。

■「吉里吉里／KAG」の嬉しい仕様

それでもまだ疑いの目で見ているお尻の重い人に、「吉里吉里／KAG」の嬉しい仕様をアナウンスします。でも、全部を書くとページが無くなるので、少尉が感心した仕様や機能の一部について、簡単に書いておくことにします。

①対応ファイル形式が豊富

画像は現在流通しているほとんどの形式が取り扱えます。また、BGMとしてはMIDIやOggVorbis*が指定できる他、音楽CDトラックの利用もできます。ミックスCD形式で配布すれば、プロ顔負けのパッケージングも不可能ではありません。また、FlashムービーやMPEGなど、新しく登場してくる主要なファイル形式についてはプラグインで随時対応しているので、エンジン自体がいつまでも陳腐化しません。

*OggVorbis
パテント・ロイヤリティー・フリーのため特許料に振り回されることのない音声圧縮形式、別名「MP3キラー」。

＊トランジション
画面を切り替えること。「ブラインド」「フェード」「渦巻き」「チェッカー」など、さまざまなトランジションがある。

②トランジション・パターンを自作できる

ビジュアル命の萌えゲームや雰囲気を大切にするアドベンチャー・ゲームでは、トランジション*もカッコよく行ないたいものですが、「吉里吉里／KAG」ではオリジナルのトランジション・パターンを何種類でもゲーム内に取り込める「ユニバーサル・トランジション」と呼ばれる機能があり、クリエイター心をくすぐります。また、「KAG」には、用い方によっては絶大な視覚演出効果を期待できる「スクロール・トランジション」と呼ばれる機能もあります。

③制限のない変数とフラグ

「吉里吉里／KAG」では、「フラグ」「数値変数」「文字変数」をほとんど同様のオブジェクトとして取り扱うことができます。このため、それぞれの利用方法を別々に学ばなくても一貫した書式で自在に活用できます。また、変数の最大数や演算に制限は一切ないので、変数資源の枯渇を気にせず自由に利用することができます。乱数生成や自由演算はもちろん、論理分岐にからめて変数の増減を行なうこともできるので、市販のアドベンチャー・ゲームに用いられているシステムを再現することも簡単です。

④デバッグが楽

玄人向けのデバッガとモニタが搭載されているので、プログラム実行時にスクリプトがどのように解釈されているか、変数がどのように扱われているかを随時確認することができます。特に変数については、プログラム実行中に恣意的に任意の値を代入することが可能なので、変数分岐処理の確認やデバッグに活用できます。慣れるまでは少し掛かるかもしれませんが、手に馴染みます。

⑤EXE形式で配布可能

完成したゲームはEXE形式にリンク（結合）して配布できます。ランタイムもプレイヤーソフトも一切不要なので、ユーザーに「何だよ。これだけじゃ動かないのかよ！　だっせー！」と失望感を味わわせることも少なく、何よりファイル1個でゲームが動いてしまうのが、市販ソフトみたいでカッコいいではありませんか。

0-2 KAGのプロジェクト・フォルダについて

「吉里吉里」のアーカイブ・ファイルには、「吉里吉里本体」の他に、吉里吉里をサウンドノベルやアドベンチャー・ゲームのエンジンとして動作させるための「KAGシステム」(以後「KAG」と表記) が含まれています。

KAGは右のようなフォルダ構造から成り立っていますが、このうちKAGの"本体"と呼べる部分は「template」フォルダ以下の10個のフォルダです。

これらは、ゲームに必要な素材を入れておくためのフォルダ群であり、これら10個のフォルダを全部まとめて「KAGのプロジェクト・フォルダ」と呼びます。

デフォルトではプロジェクト・フォルダは「template」という名前になっていますが、これはゲームのタイトルなどに自由に変更してかまいません。念のために別の場所に新しいフォルダを作り、template以下のサブフォルダをコピーして、そちらを開発に利用するとよいでしょう。

template 以下のサブフォルダには、以下のような素材ファイルを入れます。

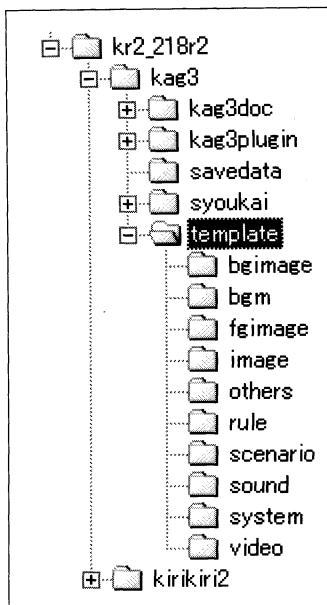
・画像を入れるフォルダ

bgimage	背景絵
fgimage	キャラクターの立ち絵など、前景に表示する絵
image	イベント絵に代表される一枚絵など
rule	トランジション用グレースケール画像ファイル

・音声や動画を入れるフォルダ

bgm	MIDI、OggVorbis形式のBGM
sound	Wave、OggVorbis形式の効果音
video	AVI、MEPG、SWF形式の動画

・文章を入れるフォルダ

scenario	KAGのシナリオテキストファイル

ワンポイント

「吉里吉里」も「KAG」も、「LZH」と呼ばれる圧縮形式で配布されていますが、これはそのままダブル・クリックしても実行できません。まずはLZHファイルを展開し、中身を取り出す必要があります。

展開するには専用のアーカイバと呼ばれるソフトウェアが必要ですが、これらはインターネットのフリーソフト紹介ページで無料ダウンロードできたり、雑誌の付録のCD-ROMなどに収録されているので、まずはそれを手に入れましょう。

なお、圧縮ファイルの取り扱いについては本書では特に説明しませんので、圧縮解凍の方法がわからない方はインターネットや雑誌の付録CDなどで別途学習してください。

・システム関係のフォルダ（追加や削除は不可）

system	KAGの設定ファイル

・その他のフォルダ

others	その他のファイル

この中でも開発中に頻繁にアクセスすることになるのが「scenario」フォルダです。

scenario フォルダには、画面に表示する文章と各種の制御を行なうタグが書き込まれる「ks」という拡張子のファイルを置きますが、KAGでゲームを制作するという行為は「このフォルダ内のシナリオ・ファイルを書く」という行為そのものに当たります。

つまり、KAGでゲームを作る場合は、素材を準備して各フォルダに配置し、後はそれらを制御するためのシナリオ・ファイルを書くという手順で行なうわけです。

■開発環境を整える

プロジェクト・フォルダである template フォルダは、そのままでは実行できません。実行するには吉里吉里本体である「krkr.eXe」が必要になります。実行方法ですが、二通りあります。

①template フォルダを「krkr.eXe」のアイコン上にドロップする
②「krkr.eXe」を起動して、実行プロジェクトにtemplateフォルダを指定する

ここで間違えやすいのは、フォルダ/アーカイブの選択時に「startup.tjs」や「first.ks、scenario」フォルダなどを指定しまうことです。これらのファイルやサブフォルダを指定してもエラーが返るだけなので、必ず template フォルダそのものを指定（またはドロップ）する必要があります。

ワンポイント

ゲームを開発している最中はプロジェクトの動作確認を行なう回数が非常に多くなるため、最初からショートカットを作っておくと便利です。

①krkr.eXe のショートカットを作成します。

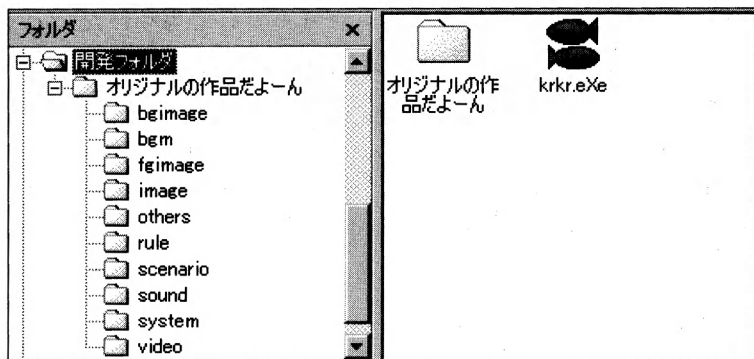
②このショートカットの「プロパティ」を開くと、[ショートカット] タブ内にリンク先という項目があります。この末尾に半角スペースを空けてから、プロジェクトフォルダのフォルダ名を、"" で囲んで書きます。

以降は、このショートカットをダブルクリックするだけで指定されたプロジェクトが吉里吉里で動作します。

プロジェクト・フォルダが吉里吉里の実行可能ファイルと違う場所にある場合はプロジェクト・フォルダ名をフルパスで指定してください。

さらにこのショートカットをスタート・メニューに登録しておけば、デスクトップ上のアイコンをダブル・クリックしなくてもキーボードの Windows キーですぐに起動できて、非常に便利です。

「krkr.eXe」は開発の都合上、プロジェクト・フォルダと同じ階層に置かれていると便利なので、まずはkirikiri2フォルダ内に置かれている「krkr.eXe」をtemplateフォルダ（名前を変えて、別に作っている場合はそのプロジェクト・フォルダ）と同じ階層にコピーしておきましょう。

動作が確認できたら、いよいよ開発を始めますが、その前に、templateフォルダ内のsysytemフォルダの中にある「Config.~new」という名前のファイルを「Config.tjs」に改名しておきます。これをせずに プロジェクト・フォルダを「krkr.eXe」で実行しようとすると、エラーになります。

プロジェクト名のついたフォルダ

0-3 Config.tjsを調整する

ソフトの定義ファイルなど普通の人は見ませんが、“KAG使い”を目指すなら、何が何でも「Config.tjs」を理解する必要があります。

ちなみに、ダウンロードしたばかりのtemplateフォルダには「Config.tjs」というファイルは存在せず、代わりに「Config.~new」というファイルがあるので、まずはこれを「Config.tjs」にリネームします。

「Config.tjs」はそのままではダブル・クリックしても開けないので、拡張子が「tjs」のファイル形式を皆さんがよく使うテキスト・エディタに関連付けておくとよいでしょう。

「Config.tjs」は、「KAG全体の設定」「ウインドウや動作の設定」「メニューの設定」「メッセージ・レイヤーの設定」「BGMの設定」「メッセージ履歴表示の設定」——の6つのエリアに分かれていますが、ここでは順を追って変更が必要な場所を解説していきます。

●全体の設定

```
// ◆ タイトル
// ウィンドウのキャプション(タイトル)および
// タスクバーに表示されるものです。
;System.title = "吉里吉里";
```

↑「吉里吉里」という文字をゲームのタイトルに直す。

```
// ◆ シナリオ解析モード
// シナリオファイル中の改行に従わせたい場合に false を設定します。
// false を指定すると、KAG のシナリオ中の改行が、行末に¥または
// [p] を書かない限りそのままメッセージレイヤ上での改行として扱われ
// ます。true を設定すると [r] タグを書かない限り改行されません。
// KAG2.x 互換にするには false を指定してください。
;global.ignoreCR = true;
```

↑本書の解説通りに演習したいのなら、「false」。

ワンポイント

WindowsMeにおけるファイルとアプリケーションの関連付けは以下のように行ないます。

まず、マイコンピュータを開き、「ツール」メニューから「フォルダオプション」を選び、「ファイルの種類」タブをクリック。さらに「新規」をクリックし、拡張子の部分にtjsと入力して「OK」をクリック。

これでtjs拡張子が登録されたので、こんどはそれをリストから探してクリックし、右下の「詳細設定」をクリック。

「新規」をクリックし、アクションに「open」と入力後、「参照」をクリックして自分がよく使うテキスト・エディタの実行ファイルを指定したら、ひたすら「OK」をクリックしてダイアログを閉じれば、設定終了です。

私はフリーのエディタ「TeraPad」を愛用していますが、この場合は「アクションを実行するアプリケーション」の部分に「TeraPad.exe」を指定した後、半角のスペースを空けてから"%1"と記述する必要があります。TeraPadを使う人は注意してください。

●ウィンドウや動作の設定

```
function KAGWindow_config()
{
// ◆ 画面サイズ
// scWidth に画面の幅、scHeight に画面の高さをピクセル単位で指
// 定します。標準的に使われている 640x480 や 800x600 のような画
// 面サイズではないサイズに設定すると、フルスクリーンにできない場合
// があります。
;scWidth = 640;
;scHeight = 480;
```

↑変更するとトラブルの元になります。

```
// ◆ 読みとり専用モード
// readOnlyMode = true とすると、ディスクへのいっさいの書き込み
// 動作をしなくなります。CD-ROM 上から実行させるときなどに指定して
// ください。もちろん、その場合は栞の保存などができなくなります。
;readOnlyMode = false;
```

↑CDで配布し、セーブさせないデモ版などは「true」に。

```
// ◆ フリーセーブ
// 栞の保存形式を指定します。
// false を指定すると、メニューバーの「栞をたどる」「栞をはさむ」の
// サブメニュー項目からセーブデータを保存したり読み込んだりします。
// この場合は保存可能な栞の最大の個数は numBookMarks によって制
// 限されます。
// true を指定するとフリーセーブモードになり、「栞をたどる」では
// 「開く」ダイアログボックス、「栞をはさむ」では「名前を付けて保存」
// ダイアログボックスが開き、栞を個別のファイルとして保存するモード
// になります。この場合は 保存可能な栞の個数に制限はありません。ま
// た、エクスプローラなどによる栞の管理が容易になります。
;freeSaveDataMode = false;
```

↑通常は「false」のままでよい。

コラム～心の底のひとりごと

たま～に変な画面サイズのゲームを見ますが、どうしてあんな中途半端なサイズで制作するのかとっても不思議だったりして。

ワンポイント

同人即売会などでCDに焼き込んだ作品を販売する人達は、特にセーブ・データの取り扱いについては注意してくださいね。
インストーラを使ってインストールさせるか、セーブ・データを保存させない指定にしたほうがトラブルが減ります。

```
// ◆サムネイルを保存するか
// false を指定すると、セーブデータの拡張子は .kdt になり、サムネ
// イルは保存されません。
// true を指定すると、セーブデータの拡張子は.bmpになり、エクスプ
// ローラや「開く」ダイアログボックスの「縮小版」表示でセーブ時点で
// のサムネイルを表示することができるようになります。
;saveThumbnail = false;
```

↑通常は「false」のままでよい。

```
// ◆ セーブデータ保存場所
// ;saveDataLocation = "savedata";
// のように指定すると、吉里吉里の実行可能ファイルと同じフォルダ以下
// のsavedata フォルダにデータが保存されます。
// ;saveDataLocation = System.personalPath + "吉里吉里の
// 栞データ";;saveDataLocation = System.appDataPath + "吉
// 里吉里の栞データ";のように指定すると、ユーザーごとのホームディ
// レクトリ以下の"吉里吉里の栞データ" というフォルダのしたに デー
// タが保存されます。この場合は、例で示した "吉里吉里の栞データ"
// の部分は他のゲームなどと重ならない、十分にユニークな名前である必
// 要があります。System.personalPath の場合は通常「マイ ドキュ
// メント」フォルダ、System.appDataPath の場合は Application
// Data になります。また、フリーセーブモードであってもシステム変数
// やシステムの設定を記録するためにこのセーブデータ保存場所の指定は
// 必須です。詳しくは KAG3ドキュメントの「セーブデータに関して」を
// 参照してください。
;saveDataLocation = "savedata";
```

↑作品の略語にすると吉里吉里で制作されたゲーム同士で重複しないため、変更すべき。

コラム～少尿のひとりごと

セーブデータ保存場所についてですが、私の拙作1999 ChristmasEveの場合はsavedatという名前になっています。何でこんな名前にしたんだろうと、今となっては不思議に思います。

```
// ◆ データ名プレフィクス
// ここで指定した名前で始まる名前で栞関連のファイルが作成されます。
// フリーセーブモードの場合でもシステム変数やシステムの設定を保存する
// ファイルに影響します。
;dataName = "data";
```

↑ここも作品の略語にする。
（例：「1999ChristmasEve」の場合は「1999xe」）

```
// ◆ 文字表示スピード (ミリ秒/文字)
;chSpeeds.fast = 10; // 「高速」文字表示スピード
;chSpeeds.normal = 30; // 「普通」文字表示スピード
;chSpeeds.slow = 50; // 「遅い」文字表示スピード
```

↑通常はこのままでよい。が、速くするとユーザーが喜ぶ。

```
// ◆ 通過記録の最大数
// 通過記録の最大数を指定します。最大、ここで指定した回数、前に戻
// る事ができます。大きくするとセーブデータも大きくなります。
;maxHistoryOfStore = 5;
```

↑一つ上の「前に戻る」機能を使わない場合は「0」でよい。

```
// ◆ 利用可能な効果音バッファの数
// 利用可能な効果音バッファの最大値を指定します。つまり、ここで指定
// した数の分だけ効果音を同時に再生できます。効果音を使用しない場合
// は 0 を指定してかまいません。
;numSEBuffers = 3;
```

↑通常はこのままでよいが、効果音を同時に2つ以上発音しなければ「1」でも足りる。

ワンポイント

効果音バッファや前景レイヤー、メッセージレイヤーなどは必要最小限の数を定義しておくとメモリの節約になり、低スペックのマシンでも動くようになります。

私の場合は、前景レイヤーは「0」、メッセージ・レイヤーは「1」の状態でConfig.tjsに定義しておき、必要になったら[laycount]タグで増やし、使い終わったら減らすようにしています。

ただし、効果音バッファについてはConfig.tjs内でしか定義できないので、注意してくだ

```
// ◆ 初期状態の前景レイヤの数
// 必要なければ 0 を指定してかまいません。
// 数が多いと速度が低下したりメモリを消費しますので、必要以上に大き
// な数を指定しない方が良いでしょう。
// laycount タグでシナリオ内でも変更できます。
;numCharacterLayers = 3;
```

↑通常はこのままでよいが、前景を使わなければ「0」にしたほうがよい。

```
// ◆ 初期状態のメッセージレイヤの数
// 前景レイヤと違って、0 を指定することはできません。これも必要な数
// だけ確保するようにすべきです。laycount タグでシナリオ内でも変更
// できます。
;numMessageLayers = 2;
```

↑通常はこのままでよいが、普通に文字を表示するだけなら「1」でもよい。

コラム～少尉のひとりごと
利用可能な栞の数についてですが、皆さんは「何でこんな大作ゲームなのにセーブ・データの保存場所がこれだけしかないんだよ！」と怒った経験はありませんか？
セーブ可能領域はあまり多すぎても緊張感がないようですが、少なすぎるのは嫌がらせ以外の何ものでもありませんよね。

```
// ◆ 利用可能な栞の数
// メニューに表示し、ユーザーが選択可能な栞の数です。
// save や load タグで保存可能な栞の数はこの設定には影響しません。
// メニューによる栞の管理を行なわない場合 ( ゲーム画面中で全部栞の
// 管理などを行なう場合など ) は、numBookMarks は必要な数に設定
// し、restoreMenu.visible と storeMenu.visible を false に
// 設定してください。
;numBookMarks = 10;
```

↑作品内容に合わせて適宜変更。そこそこ多くするとユーザーは喜ぶかも。

```
// ◆ 「ヘルプ > 目次」を選択したときに実行するファイル
// ここで指定するファイルは吉里吉里実行可能ファイルと同じ場所に設置
// されていなければなりません。
;helpFile = "readme.txt";
```

↑ヘルプ・ファイルの名前を拡張子まで含めて指定する。

●メニューの設定

```
// ◆「システム > 前に戻る」
;goBackMenuItem.visible = true;
```

↑作品内容に合わせて変更する。前に戻らせたくなければ、「false」に。

```
// ◆「文字表示 > アンチエイリアス」
;chAntialiasMenuItem.visible = true;
```

↑通常はこのままでよい。

```
// ◆「文字表示 > フォント」
;chChangeFontMenuItem.visible = true;
```

↑ユーザーにフォントを変更させたくなければ、「false」に。

```
// ◆「ヘルプ」
;helpMenu.visible = true;
```

↑通常はこのままでよい。ヘルプが無ければ「false」に。

```
// ◆「デバッグ」
;debugMenu.visible = false;
```

↑開発中は「true」に。リリースしたら「false」に。

●メッセージ・レイヤーの設定

```
function MessageLayer_config()
{
// ◆ メッセージ枠用の画像
// メッセージ枠用の画像のファイル名を指定します。
//""を指定するとメッセージ枠の画像を使用しません。また、メッセ
// ージ枠用の画像を指定した場合、position タグでメッセージレイヤ
// のサイズを変更するとおかしな表示になると思います
;frameGraphic = "";  // position タグの frame 属性に相当
```

↑メッセージ・レイヤーに枠をつけたい場合は、ファイル名を指定。

ワンポイント

「アンチエイリアス」とは、文字のギザギザ（ジャギー）を目立たなくする機能のことです。
アンチエイリアスをONにした場合、文字サイズが大きければ美しいアウトラインを得られますが、小さい文字の場合は逆に見にくくなるので、注意してください。

コラム～少年のひとりごと

いつも思うんですが、どうしてKAGの「デバッグ・モード」はデフォルトが「false」なんでしょうね。

ワンポイント

メッセージ枠を使ったサンプルがCDに収録されているので、ご参考にどうぞ。

```
// ◆ メッセージレイヤの色と不透明度
// frameColor には 0xRRGGBB 形式で ( RR GG BB はそれぞれ 2
// 桁の 16 進数)メッセージレイヤの色を指定します。frameOpacity
// には 0 ～ 255 の数値で、不透明度を指定します。メッセージ枠用の
// 画像が指定されている場合は無効です。
;frameColor = 0x000000; // position タグの color 属性に相当
;frameOpacity = 128;  // position タグの opacity 属性に相当
```

↑作品内容に合わせて色と不透明度を調整。

```
// ◆ 初期位置
// これらは、position タグのそれぞれ left top width height の
// 属性に対応します。
;ml = 16;  // 左端位置
;mt = 16;  // 上端位置
;mw = 640-32;   // 幅
;mh = 480-32;   // 高さ
```

↑この状態はノベル風の画面。アドベンチャー・ゲーム風にしたければ、「mt=400 mh=60」などに変更する。

ワンポイント

アドベンチャーゲーム風の画面についても、CDに収録されているサンプルが有効だと思います。

```
// ◆ 文字の大きさ
// デフォルトの文字の大きさ ( 高さ ) を pixel 単位で指定します。
;defaultFontSize = 24;  // deffont タグの size 属性に相当
```

↑作品内容に合わせて変更する。

```
// ◆ 行間
// 行間を pixel 単位で指定します。
;defaultLineSpacing = 6; // defstyle タグの linespacing
属性に相当
```

↑作品内容に合わせて変更する。

```
// ◆ 字間
// 字間を pixel 単位で指定します。
// デフォルトは 0 で、正あるいは負のオフセットで指定します。
// 負の数値を指定すると字間が詰まります。正の数値を指定すると時間が
// 空きます。
;defaultPitch = 0; // defstyle タグの pitch 属性に相当
```

↑作品内容に合わせて変更する。

```
// ◆ 文字の書体
// デフォルトのフォント名を指定します。
// この設定はシステム変数に記録されて、次回起動時に引き継がれるの
// で、userFace の設定を変更し、適用したい場合はシステム変数ファ
// イル?????sc.kdt の "chdefaultFace" => の行を削除してくだ
// さい。カンマで区切って複数のフォントを指定することができます。そ
// の場合は、最初の方に書いたフォントが存在すれば、優先されます。
;userFace = "ＭＳ Ｐ明朝"; // deffont タグの face 属性に相当
```

↑Windows標準のフォントを使うことが原則。

```
// ◆ 文字の色
// デフォルトの文字の色を 0xRRGGBB 形式で指定します。
;defaultChColor = 0xffffff;  // deffont タグの color 属性に
相当
```

↑作品内容に合わせて変更する。

```
// ◆ 文字をボールドにするか
// する場合は true, しない場合は false を指定します。
;defaultBold = true; // deffont タグの bold 属性に相当
```

↑作品内容に合わせて変更する。

```
// ◆ ルビサイズ
// ルビサイズを pixel 単位で指定します。
;defaultRubySize = 10; // deffont タグの rubysize に相当
```

↑作品内容に合わせて変更する。

ワンポイント

文字の書体についてですが、自分が持っているフォントを指定しても、ユーザーの中には持っていない人がいるかもしれません。普及率の少ないフォントならなおさらです。できるだけWindows標準のフォントを使うか、自分で作ったオリジナルのレンダリング・フォントを利用するようにしてください。

```
// ◆ ルビ の表示オフセット
// 負の数を指定するとその pixel 数分、本文に近い位置に表示されます。
;defaultRubyOffset = -2; // deffont タグの rubyoffset に相当
```

↑作品内容に合わせて変更する。

```
// ◆ アンチエイリアス文字描画をするか
// する場合は true, しない場合は false を指定します。
;defaultAntialiased = true;
```

↑作品内容に合わせて変更する。

```
// ◆ 影の色
// 影の色を 0xRRGGBB 形式で指定します。
;defaultShadowColor = 0x000000;  // deffont タグの shad-
owcolor に属性に相当
```

↑作品内容に合わせて変更する。

```
// ◆ 縁取りの色
// 縁取りの色を 0xRRGGBB 形式で指定します。
;defaultEdgeColor = 0x000000;  // deffont タグの edgecol-
or に属性に相当
```

↑作品内容に合わせて変更する。

```
// ◆ 影を描画するか
// する場合は true, しない場合は false を指定します。
;defaultShadow = true;  // deffont タグの shadow 属性に相当
```

↑作品内容に合わせて変更する。

```
// ◆ 縁取りをするか
// する場合は true, しない場合は false を指定します。KAG3 から、
// 縁取りと影を同時に描画することができなくなりました。縁取りを指定
// した場合は縁取りが優先されます。
;defaultEdge = false;  // deffont タグの edge 属性に相当
```

↑作品内容に合わせて変更する。

```
// ◆ リンクの強調色
// リンクを選択したときに出る半透明矩形のデフォルトの色です。
;defaultLinkColor = 0x0080ff; // link タグの color 属性に相当
```

↑作品内容に合わせて変更する。

```
// ◆ リンクの不透明度
// リンクを選択したときに出る半透明矩形の不透明度です。
;defaultLinkOpacity = 64;
```

↑作品内容に合わせて変更する。

```
// ◆ 縦書きモード
// メッセージレイヤを標準で縦書きモードにする場合は false ではなく
// true を指定してください。
;vertical = false;  // position タグの vertical 属性に相当
```

↑このままだと横書き。縦書きにしたければ「true」に。

●BGMの設定

```
function BGM_config()
{
// BGM の設定です。
// CD-DA, Wave, MIDI のいずれかを BGM として再生することができます。

// ◆ BGM再生メディア
// type 変数に設定する文字列で、使用するサウンドバッファを決めるこ
// とができます。
// "Wave"  : WaveSoundBuffer ( wav等(プラグインで再生できる
                                        物も含む) )
// "MIDI"  : MIDISoundBuffer ( mid,smf )
// "CDDA"  : CDDASoundBuffer ( cda )
;type = "MIDI";
```

↑このままだと[playbgm]で再生できるファイルは「.mid」になる。「OggVorbis」をBGMに鳴らしたければ、「MIDI」を「Wave」に変更する。

ワンポイント

BGMにMIDIファイルではなくOggVorbis形式を指定したい場合は、BGM再生メディアの指定をMIDIからWaveに変更します。
これによってOggVorbisがBGMとして使えるようになりますが、MIDIファイルは使えなくなります。

```
// ◆ CDDA再生ボリュームラベル
// "CDDA" を指定した場合は、cdVolume 変数に、再生する CD のボ
// リュームラベルを指定します。ボリュームシリアル番号ではありません。
// KAG は、このボリュームラベルを持った CD の挿入された CD-ROM ド
// ライブでCD-DA を再生します。特にマシンに複数のCD-ROMドライブ
// が装着されている場合にこれを元にしてドライブの区別を行ないます。
;cdVolume = "xxxx";
```

↑音楽CDを制御する場合は、そのCDのボリューム・ラベルを書く。

●メッセージ履歴の設定

```
// ◆ フォント名
;fontName = "ＭＳ Ｐ明朝";
```

↑作品内容に合わせて変更する。

```
// ◆ フォントを太字にするか
;fontBold = true;
```

↑作品内容に合わせて変更する。

```
// ◆ フォントのサイズ ( pixel単位)
;fontHeight = 24;
```

↑作品内容に合わせて変更する。

```
// ◆ ラインの高さ
;lineHeight = 26;
```

↑作品内容に合わせて変更する。

```
// ◆ 縦書きの場合は true
;verticalView = false;
```

↑作品内容に合わせて変更する。

ワンポイント

「メッセージ履歴」とは、過去に画面に表示された文字を読み返すための機能で、吉里吉里２の『システム』→『メッセージ履歴の表示』を選ぶか、マウス・ホイールを上に回すと見ることができます。

```
// ◆ ページ単位での閲覧を行なうかどうか
// ページ単位での閲覧をするようになると
// cm ct er タグ、あるいは repage 属性が true の hr タグ
// でメッセージ履歴の改ページを行なうようになります。
;everypage = false;
```

↑作品内容に合わせて変更する。

```
// ◆ 自動改行を行なうかどうか
;autoReturn = true;
```

↑作品内容に合わせて変更する。

0-4 「KAG」と「タグ」と「属性」

「KAG」は、「タグ」と呼ばれる命令を上から順番に並べていくだけで「吉里吉里」をゲーム・エンジンとして動作させるものです。このため、まずはこれらのタグを理解することが先決です。

タグの並べ方によっては高度な表現や演出が可能になりますし、逆に、思った通りに動作しないほとんどの理由はタグの書き間違いから生じるものです。このため、「KAGを使いこなす」ということは、「KAGに準備されているタグ使いこなす」ということになります。

タグは以下のように書きます。

●記述例

```
[er]
[image storage="test.bmp" layer=base page=fore]
```

上のタグも、下のタグも、KAGのタグです。上のタグは短くて覚えやすいですが、下のタグはいろいろ書いてあってややこしそうです。

このようにKAGには、「タグのみで使えるタグ」(上の例では「er」)と、「"属性"と呼ばれるパラメータを併記する必要があるタグ」(上の例では「image」)——の2種類があります。

「タグのみで動作するタグ」は、[]で挟んでそのまま書くだけですが、「属性をもつタグ」の場合は、以下のような書式に則って記述する必要があります。

●記述例

```
[タグ名(半角スペース)属性名=(半角イコール)属性のパラメータ]
```

属性が複数存在する場合のタグは、さらに半角スペースを空けて「属性名」と「属性のパラメータ」を半角イコールで結んだものを記述します。

なかには属性が10個近く並ぶような複雑怪奇なタグもありますが、最初はそんなものは無視してかまいません。

少しずつ自分のペースに合わせて利用できるタグを増やしていけるのもKAGの魅力です。

■タグの書き方と２種類のモード

実はKAGには２種類のタグの書き方があります。ひとつは「KAG2」までの書き方、もうひとつは「KAG3」から利用可能になった書き方です。例を見てみましょう。

・「KAG2互換モード」によるタグの書き方

```
[image storage="sample.bmp" layer=base page=fore]¥
```

・「KAG3モード」によるタグの書き方

```
@image storage="sample.bmp" layer=base page=fore
```

上の２つのタグは同じ意味です。どちらでも使いやすい方を使えばいいのですが、私は古参のKAGユーザーであることを除いてもKAG2互換モードのほうが使いやすいと思っているのですが、それはタグ行の改行処理の違いに理由があります。

「KAG2互換モード」では、タグ行の末尾に半角の「¥」を記述することで、その行を非改行にすることができます。これは言い方を変えると、「その行を、画面上では存在しない行として扱うことができる」ということになります。

たとえば、サウンドノベルの画面のように、文章が主体でその中にタグ行を記述した場合、「KAG2互換モード」では行末に半角の「¥」を置くか置かないかで、その行自体を一行の空白に当てるか、それとも存在しない行として扱うのかが決められます。

```
これは練習文です。
[playbgm storage="bgm.mid"]¥
さらに練習文が続きます。
```

上のようなシナリオを書くと、以下のように画面表示されます。

```
これは練習文です。
さらに練習文が続きます。
```

では、「¥」を外してみるとどうなるでしょうか。

ワンポイント

吉里吉里/KAG開発者のW.Dee氏の話では、古くからのKAGユーザーは[ ]で挟むKAG2互換モードを、最近のユーザーはKAG3モードを好んで利用する傾向にあるそうです。

ワンポイント

タグ行に記述する記号はすべて半角です。アルファベットはもちろん、[ ] や¥などもすべて半角で書かないと動作しません。

```
これは練習文です。
[playbgm storage="bgm.mid"]
さらに練習文が続きます。
```

上のようなシナリオの場合は、以下のように画面表示されます。

```
これは練習文です。

さらに練習文が続きます。
```

同じことを「KAG3」モードで行なおうとした場合は、次のようになります。

```
これは練習文です。
@playbgm storage="bgm.mid"
さらに練習文が続きます。
```

上のようなシナリオを書くと、以下のように画面表示されます。

```
これは練習文です。
さらに練習文が続きます。
```

では、間に一行挿入する場合はどうしたらよいのでしょうか。

```
これは練習文です。
@playbgm storage="bgm.mid"
@r
さらに練習文が続きます。
```

上の例のように[r]タグを用いて強制的に空行を挿入しなければいけません。

「KAG2モード」と「KAG3モード」を比較してみればわかりますが、行末でタグが閉じていることを視覚的に理解できる方がhtmlに似ていて馴染みやすいという理由もあるので、私は個人的にKAG2モードの方が好きです。

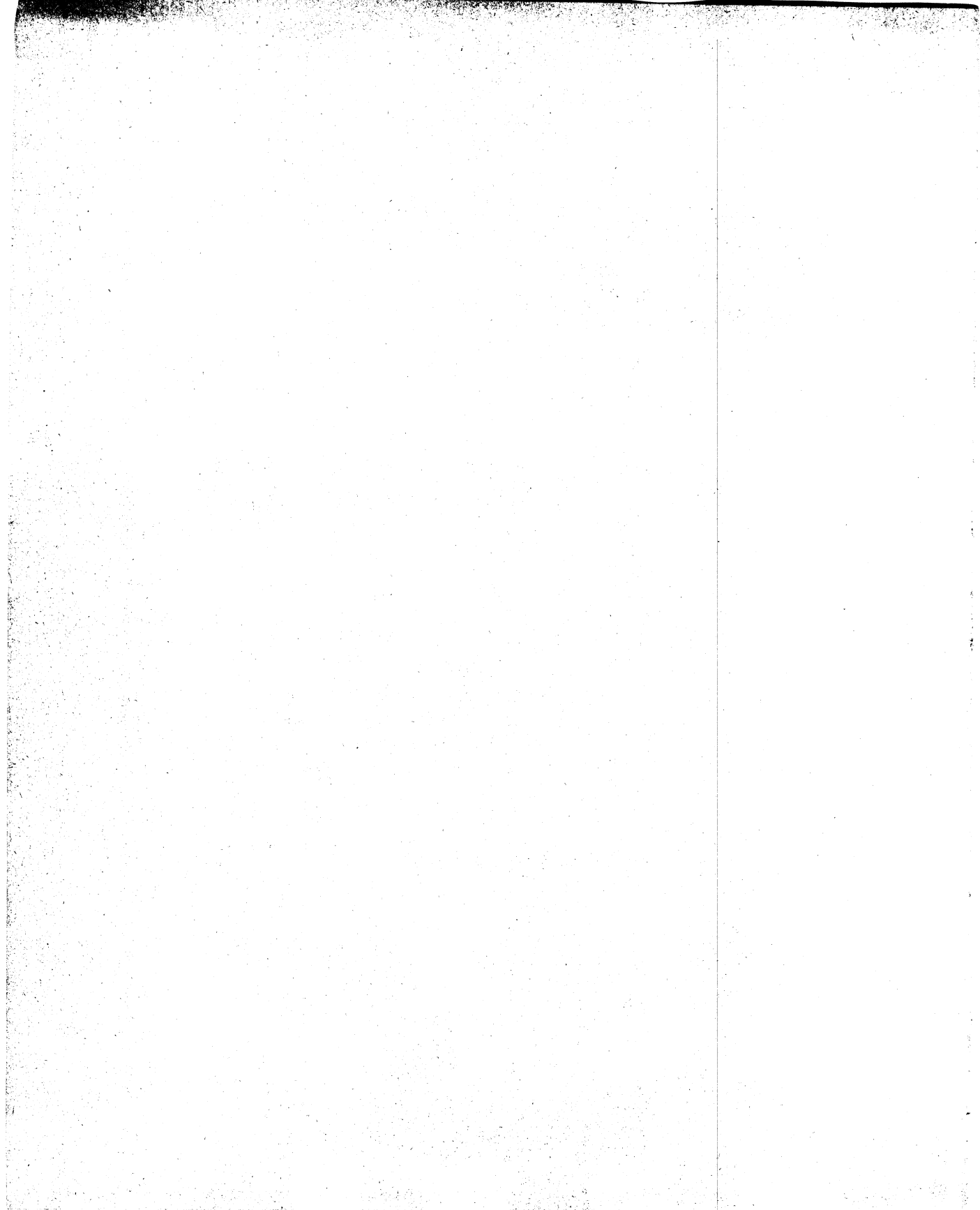
このような理由と、私自身が古くからのKAGユーザーであることから、この本では「KAG2互換モード」で説明させていただくことにします。

コラム～少豚のひとりごと

私の処女作品するある1999Christmas-Eveがリリース中断して作り直しになった直接原因が吉里吉里/KAGの存在でした。

ですので、私が吉里吉里/KAGに出会ったのは1999年の12月頃になりますね。

前項で説明した、「template」フォルダ内の「system」フォルダの中にある「Config.tjs」ファイルを変更すれば、「KAG3モード」から「KAG2互換モード」になります。

この「Config.tjs」はKAGの動作を定義するための非常に重要なファイルですが、開発者のW.Dee氏の詳しいコメントが一緒に書かれているので、KAGを動かそうとする人はぜひ読んでみてください。

Step 1 KAGの基本操作

この章では吉里吉里/KAGの基本機能を解説します。この章で学ぶ基本タグを用いれば、簡単な電子小説は作成可能になります。

1-1 文字を表示する [l][p][r][er][cm][ct]

KAGでは、文字を表示させるために必要なタグはありません。書いた文字は、それだけで画面に表示されるようになります。

「scenario」フォルダの中にある「first.ks」というファイルを開いてみると、以下のようになっています。

●シナリオ例

```
[wait time=200]
*start|スタート
[cm]
こんにちは。
```

上の例の「こんにちは」の部分が画面に表示される文字です。文字はシナリオ・ファイル内に書いていけばそのまま表示されますが、そのままだといろいろと具合が悪いので、以下のタグを用います。

[l]	改行待ち記号を表示し、クリックまたはエンター・キーの入力を待つ。
[p]	改ページ待ち記号を表示し、クリックまたはエンター・キーの入力を待つ。
[r]	強制的に改行する。

●シナリオ例

```
これを読んだら一旦停止。[l]
これも読んだら一旦停止。[l]
ここで強制的に改行[r]してみます。[l]
更に改行[r]してみます。[l]
このページはここでおしまい。[p]
```

基本的に、各行の行末には[l]*タグを用いて、次の行を表示させる前にいったん停止させます。次のクリックでページが切り替わる表現をしたい場合には、最後の行末に[p]タグを使います。

[l]タグも[p]タグも機能はほぼ同じで、画面に表示されるマークが違うだけです。これらは自由にカスタマイズできます。

ここで注意ですが、[p]タグは改ページ処理そのものを行なうわけではありません。そもそもKAGには「ページ」という概念がないので、画面をクリア

*[l]
小文字のエル

して次のページとして新たに文字を表示するためには、下記のタグのいずれかを使います。

[er]	現在の操作対象メッセージ・レイヤーをクリアし、文字属性をリセットする。
[cm]	すべてのメッセージ・レイヤーをクリアし、文字属性をリセットする。
[ct]	すべてのメッセージ・レイヤーをクリアし、文字属性をリセットした上に、操作対象レイヤーをmessage0に戻す。

画面に表示されている文字を消すタグがなぜ3種類も用意されているのかといえば、これはKAGが扱えるメッセージ・レイヤーの特性が理由になっています。

画面上に表示する文字は「メッセージ・レイヤー」と呼ばれる透明なセルのようなものの上に書かれますが、これは必要に応じて増やしたり減らしたりできます。

普通のサウンドノベルや電子小説を制作する場合は、デフォルトで指定されている1枚で足りますが、少し凝ったことをしたいと思うと、1枚では足りなくなることがあります。このような環境下で文字を消す場合に、[cm]や[ct]を使う必要が出てくるわけです。ただし、メッセージ・レイヤーを1枚しか使っていない場合は、[er]タグで充分です。

●シナリオ例

```
これを読んだら一旦停止。[l]
これも読んだら一旦停止。[l]
ここで強制的に改行[r]してみます。[l]
更に改行[r]してみます。[l]
このページはここでおしまい。[p]
[er]¥
新しいページ。[l]
さらに続く。[l]
またページを切り替えようかな。[p]
[er]¥
更に新しいページ。[l]
```

[er]タグの後ろに半角の「¥」を記述しているため、[er]の行は1行として見なされません。[er]の行を空行にしたい場合は、[er]の行末の「¥」を取ればその部分が空行になります。これでページの頭が一行だけ空きます。

1-2 背景画像を表示する [image]

文字を表示させるだけでなく、文字の背後に画像を読み込むには、[image]タグを使います。

●タグ記述例

```
[image storage="test01.bmp" layer=base page=fore]¥
```

[image] タグには他にも数多くの属性がありますが、基本となるのは上の例に出てくる3つの属性です。

「storage属性」	表示したいファイル名を書きます。フォルダや拡張子は指定しなくてかまいません。
「layer属性」	背景画像を表示するレイヤー名である「base」を指定します。
「page属性」	「fore」はフォアグラウンド、「back」はバックグラウンドの意味になります。この場合は前面である「fore」を指定します。

KAGで扱うレイヤーは、メッセージ・レイヤーであっても画像レイヤーであっても、すべてのレイヤーに裏表があります。なぜそれぞれのレイヤーに裏が必要なのかというと、画面切り替えに用いるトランジション効果(→Step2)を行なう場合、切り替わる先の画像をレイヤーの裏側に準備しておく必要があるからです。

このため、フォアグラウンドに画像をロード（page=fore）すれば画像は即座に表示されますが、バックグラウンドにロード（page=back）した場合はトランジションをしないと見えません。

●シナリオ例

```
[image storage="test01.bmp" layer=base page=fore]¥
[er]¥
背景を表示してみました。[l]
見えますか？[l]
[image storage="test02.bmp" layer=base page=fore]¥
背景だけ入れ替えてみました。[l]
ちゃんと入れ替わったかな？[l]
もういっちょ。[l]
[image storage="test01.bmp" layer=base page=fore]¥
[er]¥
１枚目に戻してみました。[l]
ついでにページも切り替えてみました。[l]
```

ワンポイント

KAGではstorage属性にファイル名を指定する時、拡張子やダブル・クオーテーションを省略できます。

ただし、後ほどファイル名で検索したり、ファイルを一括置換して別のファイルに変更したい場合もあるので、できるだけ拡張子まで含めて表記し、それらをダブル・クオーテーションマークでくくる癖をつけておいた方がよいと思います。

1-3 BGMと効果音を鳴らす[playbgm][stopbgm][fadeoutbgm][wb][playse][stopse][ws]

ゲームにBGMと効果音は必須ですが、これらはそれぞれ一つのタグを書くことによって手軽に再生することができます。

[playbgm]	BGMを再生する。
[stopbgm]	BGMを停止する。
[fadeoutbgm]	BGMをフェードアウトする。
[wb]	BGMが鳴り終わるのを待つ。
[playse]	効果音を再生する。
[stopse]	効果音を停止する。
[ws]	効果音が鳴り終わるのを待つ。

●シナリオ例

```
[er]¥
今からBGMを流してみます。[l]
[playbgm storage="test.mid"]¥
鳴っていますか？[l]
では止めてみましょう。[l]
[stopbgm]¥
いきなり止めると気分が悪いですね。[l]
もう一度鳴らしてみます。[l]
[playbgm storage="test.mid"]¥
今度はフェードアウトさせてみますね。[l]
[fadeoutbgm time=3000]¥
3000ミリ秒、つまり3秒でフェードアウトしました。[l]
更に、もう一度鳴らしてみます。[l]
[playbgm storage="test.mid"]¥
もう一度、フェードアウト。[l]
ただし、今度のは、フェードアウトが終わるまで次の文字を表示しません。[l]
[fadeoutbgm time=3000]¥
[wb]¥
はい。ご清聴ありがとうございました。[l]
[er]¥
続いて効果音のテストです。[l]
```

```
[playse storage="test.wav"]¥
鳴っていますか？[l]
しばらくしても止まりません。止めるには……[l]
[stopse]¥
これで止まったはずです。[l]
次は、一度だけ流してみます。[l]
しかも、鳴り終えるまで次の文字は表示しませんよ。[l]
[playse storage="test.wav" loop=false]¥
[ws]¥
はい。鳴り終えました。[l]
```

例のように[playbgm storage="test.mid"]と書くと、ファイルの終わりに達したら自動的に巻き戻され、再び先頭から音楽や効果音が再生されます。

ここで、オープニング・テーマやキャラクターの台詞ファイルのように一度だけ再生したい場合は、属性loopを利用します。

効果音を鳴らす[playse]タグは、デフォルトで「loop=false」となっています。つまり、効果音を１度だけ鳴らしたい場合は右のタグ記述例における「loop=false」は省略できるのです。指定した効果音を何度もループさせたい場合のみ、loop属性を用いて「loop=true」と書きましょう。

●タグ記述例

```
[playbgm storage="test.mid" loop=false]¥
```

```
[playse storage="test.wav" loop=false]¥
```

この場合、一度再生されたファイルが再生を終了するのを待って次の処理を開始したい場合がありますが、このようなときは[wl]タグを使います。

●タグ記述例

「ふざけないでよっ！」

```
[playse storage="baki.wav" loop=false]¥
[ws]¥
彼女の鉄拳がぼくの後頭部に炸裂した。[l]
```

これで音が一度鳴り終えるまで、次の文字は表示されなくなりました。

■メッセージ・レイヤーのサイズについて

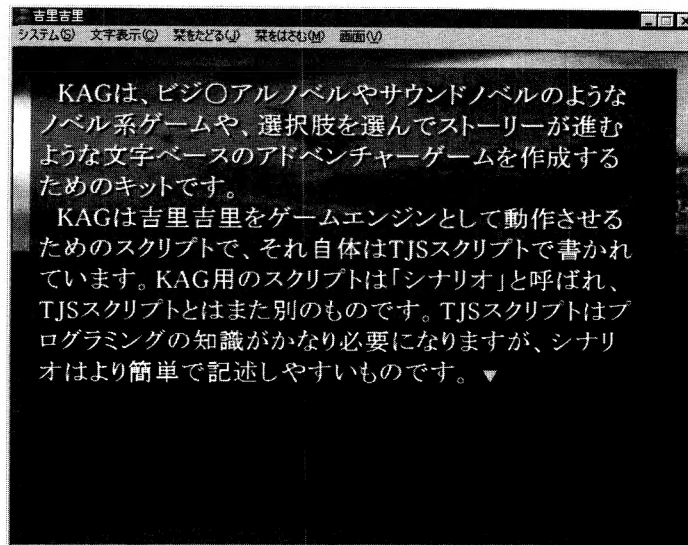
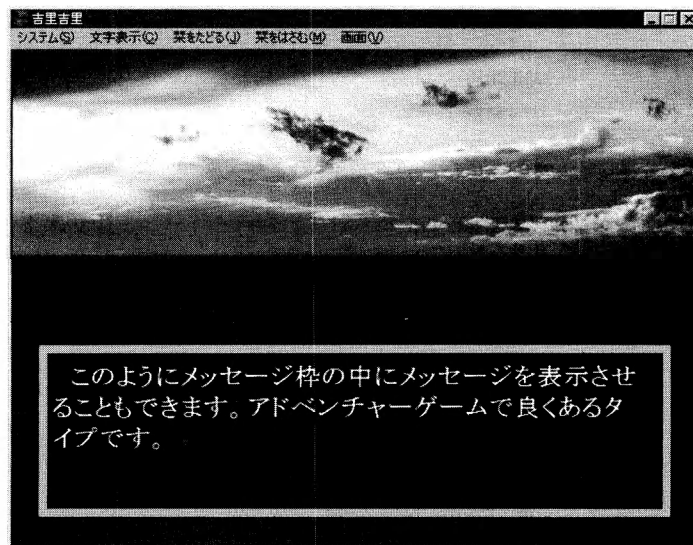
ここまでの例では、「メッセージ・レイヤー0」が画面全体に覆いかぶさるようなタイプを用いてきましたが、アドベンチャーゲームやビジュアル系のノベルゲームに見られるような、画面の下部のみに細いメッセージ・レイヤーを表示させせることもできます。

これは「Config.tjs」を書き換えることで可能になります。

●Config.tjs例

```
function MessageLayer_config()
{
// ◆ メッセージ枠用の画像
;frameGraphic = ""; // position タグの frame 属性に相当

// ◆ メッセージレイヤの色と不透明度
;frameColor = 0x660088; // position タグの color 属性に相当
↑デフォルトは「000000」で黒。ここでは「660088」にした。
;frameOpacity = 170; // position タグの opacity 属性に相当
↑透明度を調整。

// ◆ 左右上下マージン
;marginL = 20; // 左余白
;marginT = 8; // 上余白
;marginR = 8; // 右余白
;marginB = 8; // 下余白
↑文字の表示される位置を調整。

// ◆ 初期位置
;ml = 40; // 左端位置
;mt = 340; // 上端位置
;mw = 560; // 幅
;mh = 120; // 高さ
↑メッセージ・レイヤーそのものの位置を調整。
```

一例として上記のように設定した「Config.tjs」を用いると、メッセージ・

レイヤーは画面の下部に表示されるようになります。基本操作が理解できたら、作りたいゲームのデザインに合わせてメッセージ・レイヤーを変更しておくとよいでしょう。

全画面表示の例

アドベンチャー風の例

Step 2

トランジション

吉里吉里／KAGで背景画像を表示したり消したりするにはトランジションの原理を知らねばなりません。ここでは基本的なトランジションの扱いについて解説します。

2-1 トランジションの種類と原理 [trans][wt][layopt][backlay]

「トランジション」とは、メッセージ・レイヤーに書かれた文字や画像レイヤーに表示されている画像を別のものに入れ替えるときに使う機能ですが、KAGで提供されているトランジションには以下の4種類があります。

①クロスフェード・トランジション

②ユニバーサル・トランジション

③スクロール・トランジション

④拡張トランジション

この中で基本となるのは「クロスフェード・トランジション」ですが、このトランジションの原理を理解してしまえば、他の3つのトランジションについてもほぼ同様に扱えます。

なお、タグの使い方は4つのトランジションで共通なので、一度覚えてしまえば他のトランジションでも応用できます。

トランジションを行なう場合に使うタグは、[trans][wt]の二つです。[trans]はトランジションを行なうタグで、[wt]はトランジションが完全に終了するまで次の行を実行しないように終了を待たせるタグです。

ワンポイント

[ws]や[wb]と同様、処理が完全に終了するまで待つためのタグはwaitの頭文字を取った[w_]で始まることが多いようです。

●タグ記述例

```
[trans method=crossfade time=3000]¥
[wt]¥
```

トランジションの手順ですが、まずは新たに表示させたい画像を [image] タグを用いてbase（背景）レイヤーのバックグラウンドに読み込み、この後 [trans] タグを記述することによって、baseレイヤーのバックグラウンドに読み込まれた画像をフォアグラウンドに表示します。

この流れを表にすると、下のようになります。

	時間の流れ	base（裏）	base（表）
1	トランジション前	何もなし	すでに画像Aを表示中
2	トランジション準備	画像Bを読み込む	相変わらず画像Aを表示中
3	トランジション中	画像Bを表示中	画像Aが画像Bへ置き換わる
4	トランジション後	画像Bを表示中	画像Bが表示される

●シナリオ例

```
[er]¥
まずは普通に背景画像をbaseレイヤーに表示。[l]
[image storage="test01.bmp" layer=base page=fore]¥
続いて新しい画像をbaseレイヤーの裏ページへ読み込み。[l]
[image storage="test02.bmp" layer=base page=back]¥
もう読み込んだのですが、見えませんよね。裏側だし。[l]
それではいよいよトランジションします。[l]
[trans method=crossfade time=2000]¥
[wt]¥
トランジションの終了を待って、この行を表示しました。[l]
もし[wt]タグを書き忘れてしまうと、以下のようになります。[l]
[er]¥
新しい画像をbaseレイヤーの裏ページへ読み込みました。[l]
[image storage="test01.bmp" layer=base page=back]¥
読み込み完了。トランジションスタート！[l]
[trans method=crossfade time=2000]¥
書いている文字がトランジションに飲み込まれて消えちゃうよ～。消えちゃ
```

```
うよ〜。消えちゃうよ〜。[l]
[wt]¥
[er]¥
あのように消えてしまうのは変なので、trans タグと wt タグはセットで
使うことをお勧めします。[l]
```

上のサンプル・シナリオでは、メッセージ・レイヤーはそのままの状態でbaseレイヤーの画像をトランジションさせました。

ですが、画像が切り替わる瞬間をもっとはっきり見せたい場合もあると思います。このような場合は、以下のような考え方をします。

①メッセージ・レイヤーを非表示にする。
②baseレイヤーのみ表示されている状態で画像をトランジション。
③メッセージ・レイヤーを再表示する。

メッセージ・レイヤーを非表示にすれば、当然その上に書かれている文字も不可視状態になりますが、baseレイヤーの背景画像がトランジションしている間だけなので、問題はありません。

ここで利用するタグは、レイヤーの各種属性を設定する[layopt]タグです、

[layopt]には「layer」「page」「visible」その他の属性があります。この中のvisible属性を「false」にするとlayerで指定されたレイヤーが不可視になります。常用しているメッセージ・レイヤーは「message0」なので、下のように書きます。

●タグ記述例

・非表示にする場合

```
[layopt layer=message0 page=fore visible=false]¥
```

・再表示する場合

```
[layopt layer=message0 page=fore visible=true]¥
```

この記述を先ほどのサンプル・シナリオに組み込んでみます。

●シナリオ例

```
[er]¥
フォアグラウンドに画像を表示。[l]
[image storage="test01.bmp" layer=base page=fore]¥
バックグラウンドに画像を表示。[l]
[image storage="test02.bmp" layer=base page=back]¥
メッセージレイヤー0のフォア、バックともども非表示にしてから、トラン
ジションします。[l]
[layopt layer=message0 page=fore visible=false]¥
[layopt layer=message0 page=back visible=false]¥
[trans method=crossfade time=2000]¥
[wt]¥
[layopt layer=message0 page=fore visible=true]¥
メッセージレイヤー0を再表示しました。[l]
```

「メッセージ・レイヤー0」のバックグラウンドを非表示にする理由は、トランジションの原理と関わっています。

トランジションはすべてのレイヤーのバックグラウンドに存在するデータや属性をフォアグラウンドへ持ってきてしまうので、「メッセージ・レイヤー0」のフォアグラウンドをいくら不可視にしても、バックグラウンドが可視状態では意味がありません。このため、両方のページを不可視にしておく必要があるわけです。

ワンポイント

各レイヤーが持つ裏ページ（page=back）は、演劇などの舞台裏に似ています。

お客さんから見える場所（page=fore）では俳優が演技をしていて、見えない舞台裏（page=back）では役目を終えた俳優が一服したり、次のシーンの準備をしたりしているわけです。

■メッセージ・レイヤーのトランジション

[layopt]タグを使ってメッセージ・レイヤーを制御するだけでは、パッと消えたりパッと現われたりするだけで味気がないと思うかもしれませんが、当然こちらにもトランジションを用いることができます。

考え方は、非表示にしたいメッセージ・レイヤーのバックグラウンドをまず不可視にし、トランジションを行なって不可視属性のバックグラウンドを文字の書かれているフォアグラウンドに転送します。

こうして見えないものをバックグラウンドから転送したために、フォアグラウンドの文字や文字枠はトランジション効果によって消えます。

このような時に必要なのが、[backlay]タグと呼ばれるタグです。これはフォアグラウンドの属性をバックグラウンドにコピーする働きを持っています。ト

ランジション前にこのタグを用いて、全レイヤーの表画面の情報を裏画面にコピーしておきましょう

	処理手順	base(裏)	base(表)	message0.back	message0.fore
1	処理前の状態	何もなし	画像A表示中	何もなし	文章表示中
2	表画面の内容を[backlay]で裏画面へコピー	表から画像Aがコピーされたがどうせ見えない	画像A表示中	表から文章がコピーされたがどうせ見えない	文章表示中
3	message0のbackを[layopt]で不可視に	画像A表示中	画像A表示中	不可視状態になり文字と文字枠が消える	文章表示中
4	[trans]タグでトランジション	画像A表示中	画像Aが裏からコピーされるが見た目は変化なし	不可視状態	不可視状態が裏からコピーされて文字と文字枠が消える
5	処理終了	画像A表示中	画像A表示中	不可視状態	不可視状態

さらに、後半の「メッセージ・レイヤー0」の再表示についてもトランジションで行なうことができます。

	処理手順	base(裏)	base(表)	message0.back	message0.fore
1	処理前の状態	画像B表示中	画像B表示中	不可視状態	不可視状態
2	[ct] で文字レイヤーをリセット	画像B表示中	画像B表示中	不可視状態	レイヤーをリセットし、文字枠復帰
3	[backlay] で表画面の内容を裏画面にコピー	画像Bを表画面からコピーするが、同じ画像だし、どうせ見えない	画像B表示中	リセット状態の属性を表画面からコピーしてくる	リセットされた不可視状態
4	[layopt] でメッセージレイヤーの裏画面を可視状態に戻す	画像B表示中	画像B表示中	可視状態に	不可視状態のまま
5	[trans] タグで裏画面の内容を表画面へクロスフェード	画像B表示中	裏画面から画像Bがコピー	可視状態	裏画面から可視属性が表画面へロード
6	処理終了	画像B表示中	画像B表示中	可視状態	可視状態

●シナリオ例

```
[er]¥
フォアグラウンドに画像を表示。[l]
[image storage="test01.bmp" layer=base page=fore]¥
フォアグラウンドのデータと属性をバックグラウンドへコピーします。[l]
[backlay]¥
続いて、メッセージレイヤー0のバックグラウンドを不可視状態にします。[l]
[layopt layer=message0 page=back visible=false]¥
それでは、トランジション。[l]
手を触れないで待っていてくださいね。[l]
[trans method=crossfade time=3000]¥
[wt]¥
ここで文字が見えなくなりましたが、シナリオは続きます。
バックグラウンドに画像を表示して、背景のトランジション。
[image storage="test02.bmp" layer=base page=back]¥
[trans method=crossfade time=3000]¥
[wt]¥
[ct]¥
[backlay]¥
[layopt layer=message0 page=back visible=true]¥
[trans method=crossfade time=3000]¥
[wt]¥
[er]¥
メッセージレイヤー0を再表示しました。[l]
```

ただしこの例だと3秒のトランジションが3連発になってしまうので、ユーザーは1シーンの切り替えに9秒も待たされることになります。

コラム～一言のひとりごと
ウエイトばっかりのゲームはかったるいですよね。

ゲーム全編を通して一度や二度なら許されるかもしれませんが、毎回この状態が続くのは困りものです。プレイする人の気持ちを考えて、バランス良く利用するとよいでしょう。

■baseレイヤーのみのトランジション

先ほどまでの例とは逆に、こんどはメッセージ・レイヤーはまったく操作せず、baseレイヤーに表示されている画像だけを入れ替えてみます。

先にも述べたように、[trans]タグはすべてのレイヤーのバックグラウンドにあるデータや可視不可視の属性をフォアグラウンドに持ってきてしまいます。たとえそれが持ってきてほしくないものであっても、バックからフォアに、強制的に持ってきてしまいます。それを擬似的に回避するため、[backlay]のようなページ間コピー用のタグが準備されているわけです。

baseのbackに新しい画像を読み込んでトランジションさせたい場合、「message0」のbackがもし空っぽだったら、トランジションを行なうと、背景は入れ替わりますが、文字は消えてしまいます。つまり、メッセージ・レイヤーをいじらずに、背景だけ入れ替える作業をするには、ちょっとした工夫が必要になるのです。

処理手順については次のようになります。

	処理手順	base(裏)	base(表)	message0.back	message0.fore
1	処理前の状態	何もなし	画像A表示中	何もなし	文章表示中
2	裏画面の内容を[backlay]で裏画面へコピー	表から画像Aがコピーされた	画像A表示中	表から文章がコピーされた	文章表示中
3	[image]で画像Bを背景レイヤーの裏画面にロード	画像Bをロード	画像A表示中	文章表示中	文章表示中
4	[trans]タグでトランジション	画像B表示中	画像AからBへトランジション	文章表示中	backから文章がトランジションされるが同一内容のため見た目は変化なし
5	処理終了	画像B表示中	画像B表示中	文章表示中	文章表示中

このように[trans]タグでトランジションを行なうときに「メッセージレイヤー0」に変化が起きていないように見せるには、[backlay]でフォアグラウンドからバックグラウンドに同じデータをコピーしておけばいいのです。これによ

ワンポイント

実は[trans]タグにはchildrenという属性があり、children=falseとするとbaseレイヤーの内容だけがトランジションされます。

ただしこの記述を行うと演算量が膨大になり、メモリを無駄に使用するのでお勧めできません。

ってトランジション時にバックグラウンドからフォアグラウンドへデータが転送されても、結局は同じものが行ったり来たりしているだけになります。

ただ、気をつけなければならないのは、[backlay]を使うタイミングです。画像Bをロードした後に[backlay]を使ってしまうと、せっかくバックグラウンドにロードした画像Bが、[backlay]のおかげでフォアグラウンドにあった画像Aに戻ってしまいます。

トランジションが意図どおりにいかないのは、この[backlay]タグの使い方を間違えているケースが非常に多いです。

[backlay]タグが自在に扱えるようになったらKAG中級者ですね。

●シナリオ例

```
[er]¥
フォアグラウンドに画像を表示。[l]
[image storage="test01.bmp" layer=base page=fore]¥
backlayタグを使ってメッセージレイヤー0のフォアグラウンドのデータと
属性をバックグラウンドへコピーします。[l]
baseレイヤーの画像もフォアからバックへコピーされますが、次のステッ
プで新しい画像をバックにロードするので気にしなくていいです。[l]
続いて、バックグラウンドに新しい画像をロードし、トランジションしてみ
ます。[l]
[backlay]¥
[image storage="test02.bmp" layer=base page=back]¥
[trans method=crossfade time=3000]¥
[wt]¥
文字がそのままで、背景画像だけが入れ替わったように見えましたか？[l]
```

■ユニバーサル・トランジション

「ユニバーサル・トランジション」は“汎用的なトランジション”という意味で、「ルール・ファイル」と呼ばれるファイルを自作することによって、無限のトランジション・パターンを利用することができます。

クロスフェードでは、画像は単純なフェード効果によって切り替わっていましたが、ユニバーサル・トランジションはこのルール・ファイルのパターンに従って切り替わるためです。

ルール・ファイルはKAGのruleフォルダに入れて使いますが、ファイル形式はグレースケールのPNGファイルでなければなりません。

ワンポイント

vague属性にあまり大きな数値を入れるとクロスフェードと変わらなくなってしまいます。

ワンポイント

ユニバーサルトランジションを利用するときに指定する属性method=universalは省略できます。
省略するとユニバーサルトランジションが自動的に設定されたことになるからです。
method=を指定する必要があるのは、クロスフェードトランジションや後述のスクロールトランジションを利用するときだけです

●タグ記述例

```
[trans method=universal rule="rule01.png" time=3000
vague=100]¥
[wt]¥
```

属性「vague」は初めて見ると思いますが、これはユニバーサル・トランジション専用の属性です。数値は境界のあいまい値を表しており、この値が小さいとトランジションはくっきりと行なわれ、大きいとぼんやりと行なわれます。通常は100程度にしておき、シャープな切り替えをしたければ数値を下げ、ファジーにしたければ数値を上げるとよいでしょう。

●シナリオ例

```
[er]¥
背景画像をbaseレイヤーに表示。[l]
[image storage="test01.bmp" layer=base page=fore]¥
続いて新しい画像をbaseレイヤーの裏ページへ読み込み。[l]
[image storage="test02.bmp" layer=base page=back]¥
それではトランジションします。ブラインド風のトランジションです。[l]
[trans method=universal rule="rule01.png" time=3000
vague=100]¥
[wt]¥
別のルールファイルを使ってみましょうか。[l]
[er]¥
もう一度もとの画像をbaseレイヤーに表示。[l]
[image storage="test01.bmp" layer=base page=fore]¥
続いて新しい画像をbaseレイヤーの裏ページへ読み込み。[l]
[image storage="test02.bmp" layer=base page=back]¥
今度は円形のトランジション。[l]
[trans method=universal rule="rule02.png" time=3000
vague=100]¥
[wt]¥
ルールファイルのファイル名をいろいろ変更して試してみてください。[l]
```

ルールファイルは「syokai」フォルダの中に2つ入っている他、CD-ROM内に収録されているトランジション・ライブラリ（translib2.lzh）にもたくさん入っています。

最初のうちはライブラリのファイルを利用しておいて、表現力が不足してきたら自作するとよいでしょう。

■スクロール・トランジション

「スクロール・トランジション」は、その名の通り画像をスクロールさせて表示するためのものです。

●タグ記述例

```
[trans method=scroll time=3000 from=left stay=stayfore]¥
[wt]¥
```

from属性は方向を表わします。「left」「right」「top」「bottom」の4種類があるので、演出効果を考えて方向を指定します。

また、スクロール・トランジションは1種類のトランジションですが、stay属性を変更することによって3種類の表現ができるようになっています。

- **stay=stayfore**
  表画面の画像の上に、裏画面の画像がfromで指定した方向から入ってくる。

- **stay=stayback**
  表画面の画像がfromで指定した方向へ出て行き、裏画面の画像がその下から見えてくる。一般の紙芝居の形式。

- **stay=nostay**
  表画面の画像が出て行くと同時に裏画面の画像が入ってくる。このため二枚の画像が連結されて動いているように見える。

特に演出効果として優れているのが、「stay=nostay」で利用した場合です。

画面の2倍のサイズに該当する横長または縦長の画像を用意し、それを半

分に切って２枚に分けて準備しておけば、スクロール・トランジションを用いることによって視点を移動させるような効果を出すことができます。

●シナリオ例

```
[er]¥
フォアグラウンドに画像を表示。[l]
[image storage="scrolltest01.bmp" layer=base page=fore]¥
見えませんけど、バックグラウンドに画像を表示。[l]
[image storage="scrolltest02.bmp" layer=base page=back]¥
スクロールトランジション。[l]
[layopt layer=message0 page=back visible=false]¥
[layopt layer=message0 page=fore visible=false]¥
[trans method=scroll time=3000 from=bottom stay=nostay]¥
[wt]¥
[layopt layer=message0 page=fore visible=true]¥
視点の移動効果が出ていますか？[l]
```

ワンポイント

トランジションの種類と必要な属性について整理してみると、以下のようになります。

クロスフェードトランジション
method　time

ユニバーサルトランジション
rule　time　vague

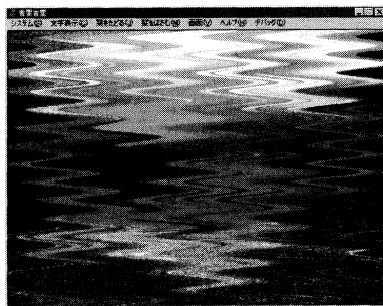
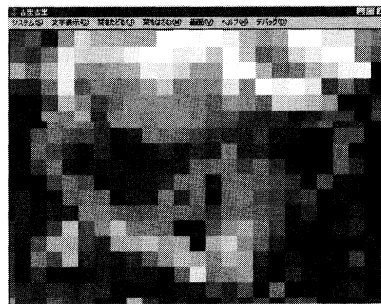
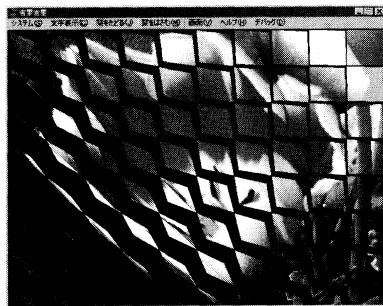
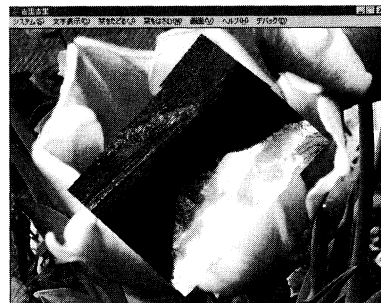
スクロールトランジション
method　time
form　stay

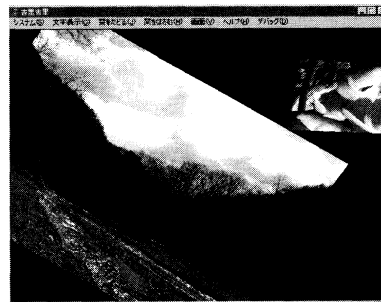
2-2 拡張トランジション

「拡張トランジション」は、吉里吉里のプラグインを利用することによって行なうことが可能なトランジションです。

利用するためには、krkr218フォルダ内にあるpluginフォルダの中の「extrans.dll」ファイルを、「krkr.eXe」と同じ場所にコピーする必要があります。

現在のところ、拡張トランジションには6種類のハンドラが準備されています。これらを切り替えることによって、さまざまなトランジションが利用できます。

wave ハンドラ

mosaic ハンドラ

turn ハンドラ

rotatezoom ハンドラ

rotatevanish ハンドラ

rotateswap ハンドラ

ワンポイント

プラグインとは、本来のプログラムとは別の状態で配布されている追加プログラムのことです。

これを本来のプログラムに追加することによって、本来のプログラムが持つ機能を拡張することができます。

要するに、ロボットの拡張パーツみたいなものですね。空を飛ぶ羽とか、地中を進むドリルとか。

拡張トランジション機能はプラグインで提供されていますから、KAGシナリオの先頭にプラグインをロードする1行を書く必要があります。

●タグ記述例

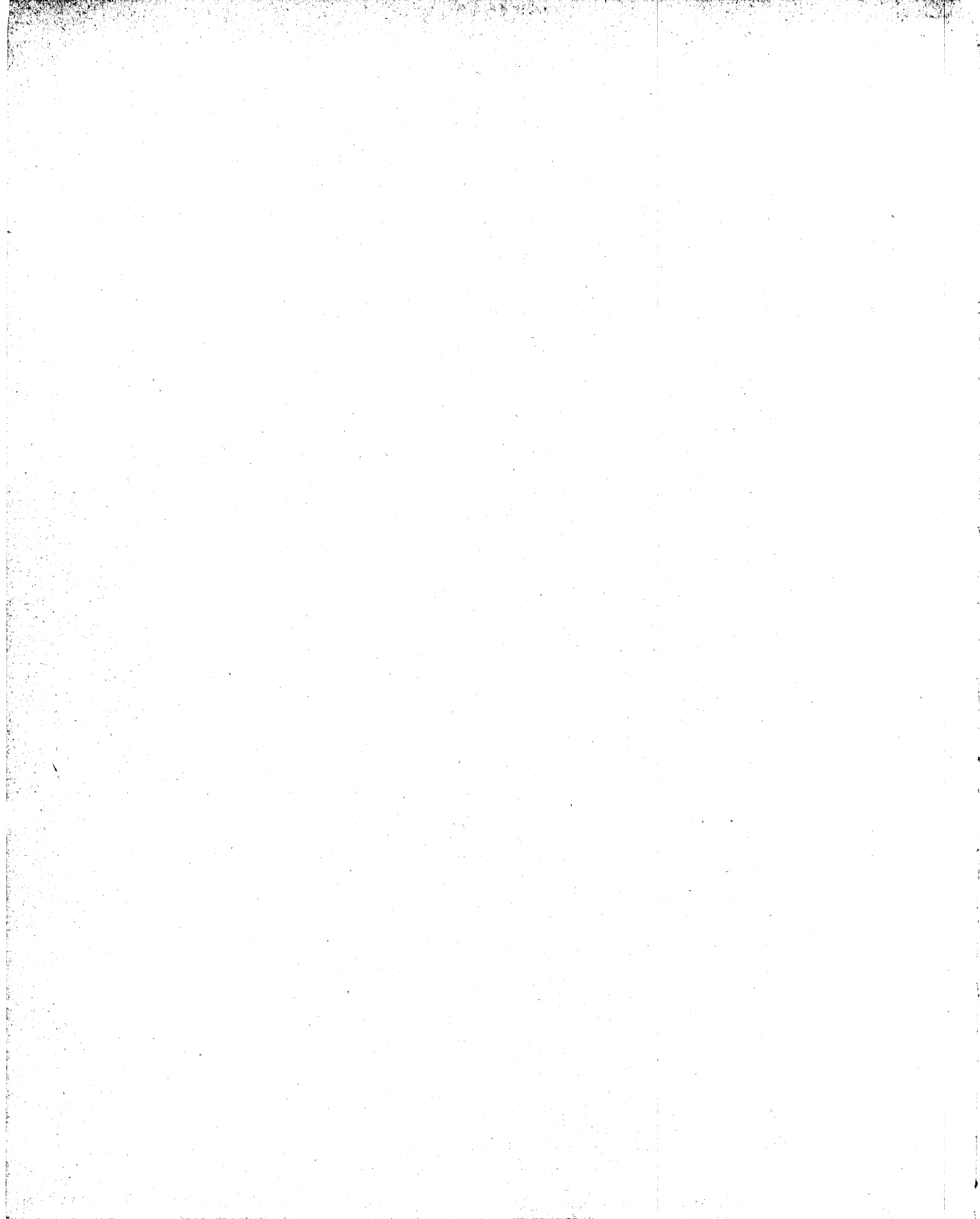
```
[loadplugin module="extrans.dll"]¥
```

●シナリオ例

```
[er]¥
画像をbaseのforeに表示。[l]
[image layer=base page=fore storage="test01.bmp"]¥
画像をbaseのbackに表示。[l]
[image layer=base page=back storage="test02.bmp"]¥
まずはwaveハンドラによるトランジション。[l]
[trans method=wave time=3000]¥
[wt]¥
[er]¥
次のトランジション用に画像をリセット。[l]
[image layer=base page=fore storage="test01.bmp"]¥
[image layer=base page=back storage="test02.bmp"]¥
次はmosaicハンドラによるトランジション。[l]
[trans method=mosaic time=3000]¥
[wt]¥
[er]¥
次のトランジション用に画像をリセット。[l]
[image layer=base page=fore storage="test01.bmp"]¥
[image layer=base page=back storage="test02.bmp"]¥
次はturnハンドラによるトランジション。[l]
[trans method=turn bgcolor=0x000000 time=3000]¥
[wt]¥
[er]¥
次のトランジション用に画像をリセット。[l]
[image layer=base page=fore storage="test01.bmp"]¥
[image layer=base page=back storage="test02.bmp"]¥
次はrotatezoomハンドラによるトランジション。[l]
```

ワンポイント

拡張トランジションをもっと利用したい人は、kr2docフォルダ内の吉里吉里2のドキュメントの中にある『トランジションについて』をご覧ください。
各ハンドラの属性について細かい情報が掲載されています。

```
[trans method=rotatezoom factor=0 time=3000]¥
[wt]¥
[er]¥
次のトランジション用に画像をリセット。[l]
[image layer=base page=fore storage="test01.bmp"]¥
[image layer=base page=back storage="test02.bmp"]¥
次はrotatevanishハンドラによるトランジション。[l]
[trans method=rotatevanish time=3000]¥
[wt]¥
[er]¥
次のトランジション用に画像をリセット。[l]
[image layer=base page=fore storage="test01.bmp"]¥
[image layer=base page=back storage="test02.bmp"]¥
次はrotateswapハンドラによるトランジション。[l]
[trans method=rotateswap time=3000]¥
[wt]¥
これで終わりです。[l]
```

Step 3

前景レイヤーを使ったキャラクター表示

キャラクターの立ち絵やシルエットなどの画像を読み込むには前景レイヤーを利用します。ここでは前景レイヤーの使い方と、立ち絵のアニメーションについて解説します

■前景画像を表示する

KAGでは1枚の背景画像の他に、任意の数の前景を表示させるための「前景レイヤー」と呼ばれるレイヤーがあります。これらの前景レイヤーには、主としてキャラクター・シルエットや立ち絵などの、背景の上に乗せて利用する絵を読み込んで利用します。

前景レイヤーと背景レイヤーの違いは以下の通りです。

比較項目	前景レイヤー（0,1,2）	背景レイヤー（base）
枚数	0〜3枚（論理上は∞）	1枚
初期状態	非表示	表示
画像の移動	可	不可

ワンポイント

前景レイヤーは論理上無制限の数を扱うことができますが、実質的にはPCのメモリやリソースなどに依存するため、1000枚も2000枚も使うことはできません。
ですが、小さな前景レイヤーなら30枚や50枚程度使用しても特に問題はないので、一度見た画像をサムネイル表示するCGアルバムなどに利用することができます。

前景レイヤーに画像を表示する方法は背景レイヤーの時とほぼ同じですが、前景レイヤーを指定した時のみ利用できる属性が2つ追加されます。

●タグ記述例

```
[image storage="charatest01.bmp" layer=0 page=fore
left=200 top=0]¥
```

top属性は前景レイヤーに読み込む画像ファイルの上端位置を、left属性は左端位置を指定するために使います。この属性を省略すると、画面の左上を基点として前景画像が表示されます。
また、前景レイヤーは最初は非表示になっているので、前景レイヤーを利用する前に必ず[layopt]タグを用いて前景レイヤーを利用可能にします。

●タグ記述例

```
[layopt layer=0 page=fore visible=true]¥
[layopt layer=0 page=back visible=true]¥
```

上の記述例では、前景レイヤー0のフォアグラウンドとバックグラウンドの両方を可視状態にしていますが、必要に応じて片方だけ利用してもかまいません。

●シナリオ例

```
[er]¥
背景を表示します。[l]
[image storage="test01.bmp" layer=base page=fore]¥
前景レイヤー0のフォアグラウンドを可視状態にします。[l]
[layopt layer=0 page=fore visible=true]¥
左上に黒い四角が見えるのは、前景レイヤーを可視状態にした時に何も画像
が表示されていないことを警告するためのものです。[l]
画像を読み込めば消えるので、前景のキャラ絵を前景レイヤー0のフォアグ
ラウンドに表示してみましょう。[l]
[image storage="akira.png" layer=0 page=fore top=0
left=200]¥
[er]¥
表示されました。[l]
続いてトランジションで表示してみます。[l]
前景レイヤーを開放します。[l]
[freeimage layer=0]¥
[er]¥
[image storage="test02.bmp" layer=base page=fore]¥
[backlay]¥
背景をbaseレイヤーのフォアグラウンドに表示し、ついでにバックグラウ
ンドにもコピーしておきました。[l]
続いて、前景レイヤー0のバックグラウンドを可視状態にします。[l]
[layopt layer=0 page=back visible=true]¥
前景のキャラ絵を前景レイヤー0のバックグラウンドに表示します。[l]
[image storage="akira.png" layer=0 page=back top=0
left=200]¥
見えませんよね。[l]
それでは、トランジション。[l]
[trans method=crossfade time=3000]¥
[wt]¥
トランジションで前景が表示されました。[l]
```

前景に利用する画像ファイルは、ファイル容量が大きくなってもよいのなら、展開速度の速い、32bitBMPを使い、配布形態の関係などから、容量を抑えたいのであればαチャンネル付PNGを使うとよいでしょう。

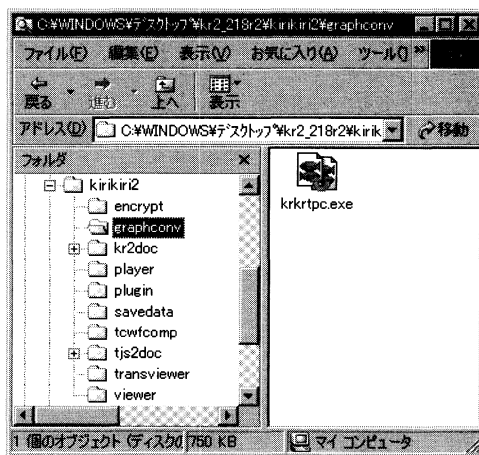
前景画像はキャラクターの立ち絵やシルエットですから、色が置かれている部分以外の部分は透明になっていないとゲーム内で使えません。

このため、前景画像を作る場合にはいくつか注意しなければならないことがあります。

まずは、グラフィック・ソフトで絵を描く時に、背景の部分に何も色を乗せず、あくまでも透明の状態で書くことです。

作成したファイルはαチャンネルと共に保存しますが、この状態ではまだKAGでは読み込めない場合もあります。

そこで利用するのが、kirikiri2フォルダ内の「graphconv」の中にある、「krkrtpc.exe」というプログラムです。

これをダブル・クリックで起動し、前景画像として利用したいファイルをウインドウの上にドロップすれば、KAGで読むことのできるファイル形式に変換して出力してくれます。

■前景画像のアニメーション

前景レイヤーに表示させた画像は、もしそれがアニメーション形式のものであれば自動的にアニメーションを行ないます。

例をあげると、美少女系のアドベンチャーゲームにある目パチや口パクなどですが、ファイルを準備しておけばKAGの方で勝手にアニメーションを実行してくれるので、シナリオへの記述については通常の前景画像を読み込む書き方とまったく同じです。

前景レイヤーに表示させるキャラクターをアニメーションさせる場合に必要なものは3つあり、ベースになる画像ファイル、アニメーションする部分のみを集めたアニメーション用画像ファイル、そしてアニメーションの秒数やコマを指定するASDファイルです。これら3つのファイルは、以下のように名前の付け方が決まっています。

ファイルの種類	名前の付け方	サンプル例
ベースとなる画像ファイル	任意のファイル名.png	akira.png
アニメーションファイル	任意のファイル名_a.png	akira_a.png
ASDファイル	任意のファイル名.asd	akira.asd

これらのファイルは3つまとめて同一のフォルダに置いておく必要があります。フォルダはどこでもかまいませんが、管理のしやすさを考えた場合、「fimage」や「other」がよいでしょう。

ASDファイルの中は、以下のようになっています。

```
*go
@loadcell
@loop
;
@macro name=copyone
@copy dx=62 dy=179 sx=%x sy=0 sw=90 sh=55
@endmacro
;
*start
@copyone x=0
@wait time=100
@copyone x=90
```

```
@wait time=100
@copyone x=180
@wait time=70
@copyone x=90
@wait time=70
@copyone x=0
@wait time=3000
;
@jump target=*start
```

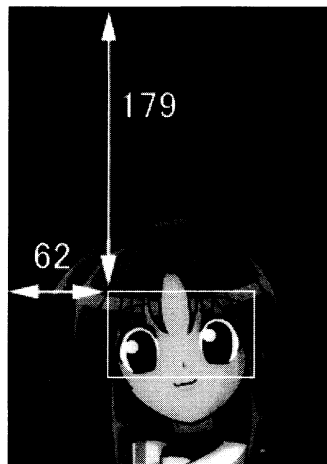
ASDの内容はサンプル例ですが、必要に応じて「=」の後ろにある数値を修正してください。それぞれの数値の意味は、以下のようになります。

●「@copy dx=62 dy=179 sx=%x sy=0 sw=90 sh=55」について

```
dx=62  dy=179
```

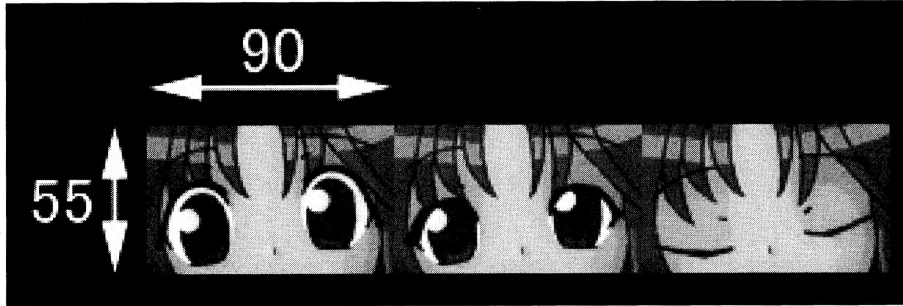
これはアニメーション用の画像をベース画像の左上から何ピクセルの位置にアニメーションファイルのパターンセルを表示するかを指定する値です。「dx」が横ピクセル数で、「dy」は縦ピクセル数です。

両方とも「0」にするとアニメーションパターンがベース画像の左上に表示されます。また、数値をベース画像のサイズ以上に指定するとベース画像の外に出てしまい、表示されなくなります。

ワンポイント
KAGシナリオ上では位置合わせが非常に難しいので、画像処理ソフト上でピクセル単位の位置決めをしてからKAGに持ってきたほうがいいでしょう。

```
sx=%x sy=0
```

これはアニメーション用の画像から1枚ごとのセルを切り出す位置を指定する値です。ここではマクロを使って表記されているのですが、意味は分からなくても数字だけ修正すればすぐに利用できます。(マクロ→Step7)

```
sw=90 sh=55
```

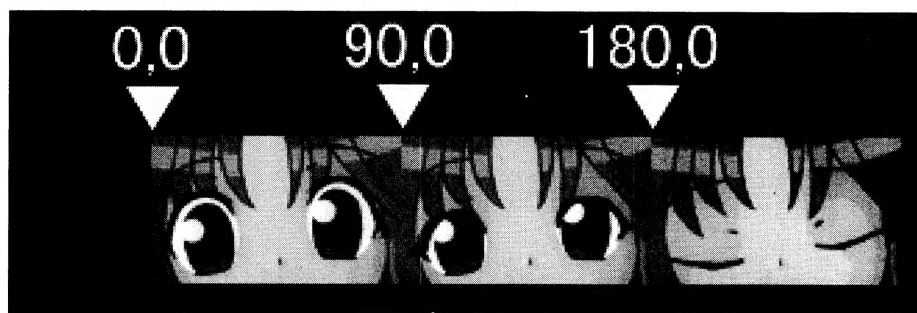
これはアニメーション用の画像に含まれているセルの1つあたりのサイズを指定する値です。サンプルでは90ドット×55ドットのセルが3枚並んで270×55ドットのアニメーション・ファイルになっていますが、指定するのは1枚の大きさですから「90」と「55」になります。

●その他の部分について

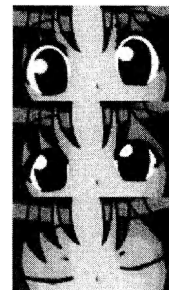
```
*start
@copyone x=0
@wait time=100
@copyone x=88
@wait time=100
@copyone x=176
@wait time=70
@copyone x=88
@wait time=70
@copyone x=0
```

この中から「@copyone x=」の行だけを抜き出して、xの値を先ほどの(sx,sy)の部分に代入してみると、

```
sx=0 sy=0
sx=90 sy=0
sx=180 sy=0
sx=90 sy=0
sx=0 sy=0
```

となります。これは何をしているかというと、アニメーション・ファイルの1枚ごとのセルがどの位置から始まるのかを指定しているのです。

1枚目のセルには、アニメーション・ファイルの左上を原点として (0,0) の位置から「sw=90 sh=55」で指定されたぶんの画像が当てられます。**2枚目**のセルには、アニメーション・ファイルの左上を原点として (90,0) の位置から「sw=90 sh=55」で指定されたぶんの画像が当てられます。**3枚目**のセルには、アニメーション・ファイルの左上を原点として (180,0) の位置から「sw=90 sh=55」で指定された分の画像が当てられます。**4枚目**のセルには、2枚目のセルで選択した画像がそのまま当てられます。**5枚目**のセルには、1枚目のセルで選択した画像がそのまま当てられます。

ここまでで、目を開いている状態から一度閉じ、さらにまた開くまでのサイクルがアニメーションとして記述できたことになります。

ここまでの例のように、横にアニメーションのセルパターンを配置する場合は、「sy」の値は常に「0」になりますが、右のような縦置きのアニメーションパターン・ファイルを利用する場合は、逆に「sx」が常に「0」で、「sy」をマクロで変化させていく必要があります。

●ASDファイル例

```
*go
@loadcell
@loop
;
@macro name=copyone
```

```
@copy dx=62 dy=179 sx=0 sy=%y sw=90 sh=55
@endmacro
;
*start
@copyone y=0
@wait time=100
@copyone y=55
@wait time=100
@copyone y=110
@wait time=70
@copyone y=55
@wait time=70
@copyone y=0
@wait time=3000
;
@jump target=*start
```

最後に[wait]タグで指定された秒数ですが、これはアニメーションの次セルを表示するまでに待つ時間を指定します。抜き出して並べてみると、以下のようになります。

```
@wait time=100
@wait time=100
@wait time=70
@wait time=70
@wait time=3000
```

開いている状態から閉じる状態までは0.1秒刻み、閉じている状態から開いた状態までは0.07秒刻みにしてあります。

いちばん下にある[wait]タグは、瞬きをしてから次に瞬きをするまでの間隔を指定するもので、ここでは3秒にしてあります。

コラム〜少年のひとりごと

まぶたは下ろす速度よりも上げる速度を少しだけ早くしたほうが自然だとW.Deeさんから教わりました。彼はこのようなことをよく観察しているのでしょうか??

ワンポイント

パソコンのやりすぎでまばたきの回数が減り、目が乾いてしまう「ドライアイ」という症状のキャラクターの場合は、最後の[wait]で指定する時間を60秒くらいにしておくといいかもしれませんね。

■前景画像の移動[move][wm]

前景レイヤーに表示させた画像は移動させることができます。このような場合は[move]タグを使います。通常は移動が完全に終了してから次の命令を行なわせることが多いため、必要ならば移動の終了を待つ[wm]タグも使います。

●タグ記述例

```
[move layer=0 time=3000 path="(640,0,255)"]¥
[wm]¥
```

「layer属性」では移動させる画像が表示されているレイヤー名を指定するので、この場合は「前景レイヤー0」を指定します。

「time属性」には移動を行なう時間を指定します。

「path属性」には、移動先の（X座標、Y座標、レイヤーの濃度）をそれぞれ指定します。

「X座標」の値は、画面全体の左上を「0」として、表示されている画像の左端が右側に何ピクセルの位置まで移動するかを指定します。後述のシナリオ例の場合は画面の右端を指定したので、画像は画面の外へ向かって出て行くことになります。

「Y座標」も同様ですが、こんどは縦位置です。画面全体の左上を「0」として、表示されている画像の上端が下側に何ピクセルの位置まで移動するかを指定します。後述のシナリオ例の場合は「0」なので、垂直方向へは移動しないことになります。

「レイヤー濃度」は、前景レイヤーの透明度を操作する値です。「255」なら何も変わりませんが、「0」にすると完全に透明になります。中間値を設定すると半透明になります。

ワンポイント

KAGでタグを記述する場合は、「"」や「'」の数が偶数でないとエラーになります。

「"」や「'」は何かをクオートする（括る）ために用いる記号なので、奇数の場合は終わりが閉じていないことになるからです。

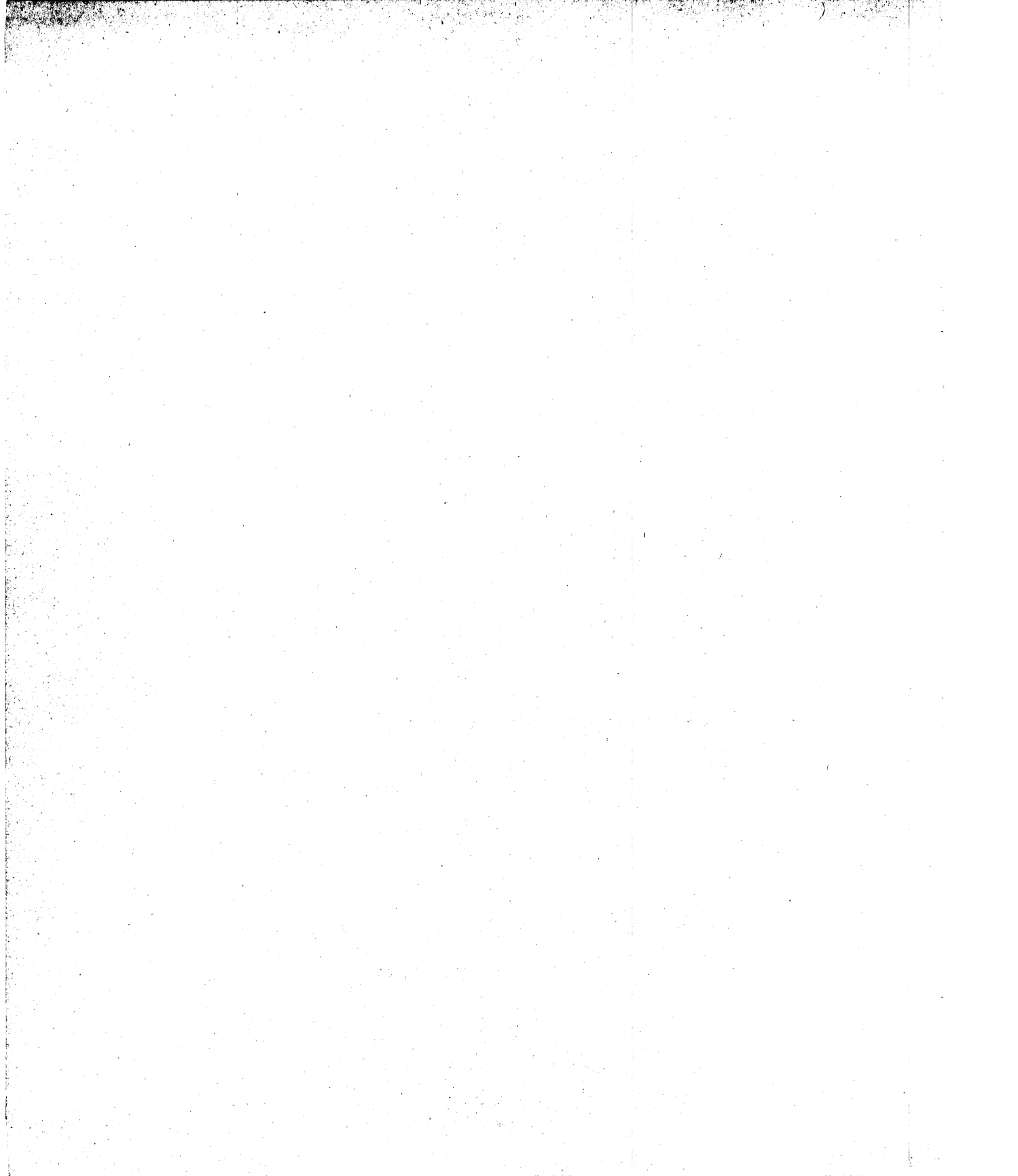
変数や条件分岐を扱う上で、思ったように動かずエラーが出るのは、このクオーテーションマークが対になっていない場合がほとんどです。

●シナリオ例

```
[er]¥
背景を表示します。[l]
[image storage="test01.bmp" layer=base page=fore]¥
前景レイヤー0のフォアグラウンドを可視状態にしてから、前景のキャラ絵
を前景レイヤー0のフォアグラウンドに表示してみます。[l]
[layopt layer=0 page=fore visible=true]¥
[image storage="akira.png" layer=0 page=fore top=0
left=200]¥
表示されました。[l]
では、右のほうへ移動させてみましょう。[l]
[move layer=0 time=3000 path="(640,0,255)"]¥
[wm]¥
移動しました。[l]
[er]¥
もう一度背景を表示します。[l]
[image storage="test02.bmp" layer=base page=fore]¥
もう一度前景レイヤー0にキャラ絵を表示。[l]
[layopt layer=0 page=fore visible=true]¥
[image storage="akira.png" layer=0 page=fore top=0
left=200]¥
表示されたので、今度は下のほうへ移動させてみます。[l]
しかも、最終的には透明になるようにしてみます。[l]
[move layer=0 time=3000 path="(200,480,0)"]¥
[wm]¥
消えてしまいました。[l]
```

ワンポイント

左のシナリオ例は、画面サイズを標準の640×480で動作させた場合の例です。

画面の外にキャラクターを移動させたい場合は、自分が扱っている画面サイズがいくつなのかによって、指定する数値が異なってきます。

Step 4

文字表示のテクニック

画像だけでなく文字表示においても細かい調整が行なえる点は吉里吉里／KAGの魅力です。ここでは文字表示に関するテクニックを解説します。

4-1 フォント属性による文字装飾 [font][resetfont]

KAGシナリオに書かれた文字は、「Config.tjs」内で指定された色や大きさで画面に表示されますが、これは[font] タグを使うことによってシナリオ中で任意に変更することができます。

●タグ記述例

```
[font color=0xff0000 size=25 face="ＭＳ 明朝"]
[font color=default size=default face=default]
[resetfont]
```

シナリオ内に[font]タグが書かれると、それより後の部分がタグで指定された文字属性に従うようになります。

色や大きさを元に戻したい場合は、２行目の[font color=default....]のような書き方をして各属性を「default」に戻すか、すべての属性を一気に「Confit.tjs」で指定された値に戻す[resetfont]タグを使います。

２行目のような書き方は、一部戻したくない属性がある場合などに利用しますが、[font][resetfont]をペアで利用したほうが、htmlの「～」のような認識ができるため、理解しやすいと思います。

ワンポイント

color属性に指定されている0xff0000は、色をRGBの16進数で表したもので、この場合は赤色を指定しています。

これらのカラーコードはhtmlのものとまったく同じですから、手持ちのhtmlエディタのカラーパレットから16進数表記を確認すれば、どの色にどの数字を使うのかが簡単に分かります。

ワンポイント

size属性に指定される数値はピクセル単位です。htmlのフォント指定パラメータとは異なるので注意してください。

●シナリオ例

```
洞窟を進んでいくと、足に何か当たった。[l]
ゆっくりと懐中電灯の光を向けてみる。[l]
光の中に浮かび上がったのは、変色した[font color=0xff0000]頭蓋骨
[resetfont]だった。[l]
[font size=52]「うわあ！」[resetfont][l]
ぼくは大声を上げると逃げ出した。[l]
```

実行画面

4-2 文字の表示速度[delay][nowait][endnowait]

KAGシナリオに書かれた文字は、「Config.tjs」内で指定された速度で表示されますが、これは[delay]タグを使うことによってシナリオ中で任意に変更することができます。

●タグ記述例

```
[delay speed=500]
```

「speed属性」には、文字を表示させる間をミリ秒単位で指定します。500の場合は、1文字表示するごとに0.5秒待つことになります。

これも前述の[font]タグと同様に、タグが書かれた位置からタグで指定された速度に従って表示されます。

元に戻す場合は「speed=user」としますが、これはユーザーがプレイ時に吉里吉里のメニューで文字の表示速度を変更している場合があるからです。

●タグ記述例

```
[delay speed=user]
```

意味としては「speed=default」なのですが、その規定値がuserに委ねられているため、このような表記になります。

また、ウエイトを「0」にして一気に表示させる場合は、speed属性に以下のように指定します。

●シナリオ例

```
まずは普通に。[l]
[delay speed=nowait]¥
ここだけ一気に表示する。[l]
[delay speed= user]¥
元に戻しました。[l]
```

これは、[nowait]タグを用いて次のように書くこともできます。

コラム～少藤のひとりごと

多くの場合、ユーザーは最高速でメッセージを表示させていることが多いようです。ちなみに私もそうなのですが、どうして市販のゲームはあんなにメッセージ表示速度が遅いんでしょうね。まさかクリア時間を意図的に引き延ばすため……？

ワンポイント

私の場合は、速度を細かく調整する場合は[delay]タグを、一気に表示させる場合は[nowait]タグを使っています。

●シナリオ例

```
まずは普通に。[l]
[nowait]¥
ここだけ一気に表示する。[l]
[endnowait]¥
元に戻りました。[l]
```

ウエイトを掛けないのなら、こちらのタグのほうが扱いやすいと思います。

■文字の表示ウエイト[wait]

文字を表示しているときに、次の文字を表示するまでに一定の時間待たせたいことがありますが、このような場合は[wait]タグを用います。

●タグ記述例

```
[wait time=500]¥
```

●シナリオ例

```
「どういうことなのよ！」[l]
[wait time=1000]¥
「じ……[wait time=1000]実は……[wait time=1000]これには深いわけが……」[l]
「もっとハキハキしゃべりなさいよ！」[l]
```

実は、この[wait]タグはクリック連打による読み飛ばしができてしまいます。演出の都合上どうしても待たせたい場合には、属性canskipを使って、以下のように書きます。

ワンポイント

作り手と遊び手の意識の差がいちばん出るのが、このウエイト処理ですね。

「遅いぞー遅いぞー」と言いながらボタンやキーを押しまくった経験は誰にでもあると思いますが、このあたりはできるだけユーザーフレンドリーな仕様にしてあげましょう。

●タグ記述例

```
[wait time=500 canskip=false]¥
```

このように書かれた[wait]タグは、ユーザーが何をどうしようと必ず指定の時間だけ待つようになります。あまり多用すると嫌がらせになるので、注意してください。

4-3 文字の位置表示[locate][style][resetstyle]

KAGシナリオに書かれた文字は、「Config.tjs」内で指定された「メッセージレイヤー0」上に順番に表示されますが、[locate]タグを用いることによって任意の場所に表示することができます。

●タグ記述例

```
[locate x=100 y=200]¥
```

また、センタリングや左詰、右詰などをしたい場合は[style][resetstyle]タグを使います。

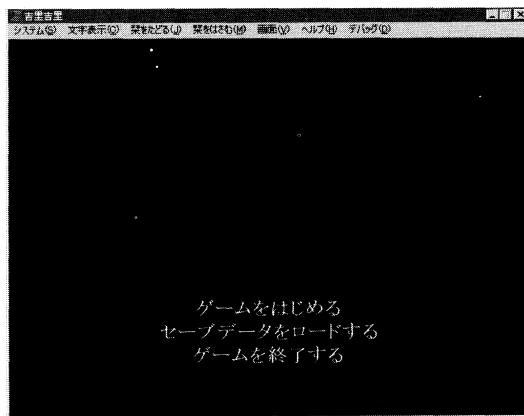
●タグ記述例

```
[style align=center]¥
[resetstyle]¥
```

align属性は、「横書き」の場合は「left」「center」「right」のいずれかを指定します。「縦書き」の場合は、「top」「center」「bottom」のいずれかを指定します。

位置を元の状態に戻す場合は、align属性に「default」を指定するか、[resetfont］タグを使います。

●シナリオ例

```
*start
[locate y=300]¥
[nowait]¥
[style align=center]¥
ゲームをはじめる
セーブデータをロードする
ゲームを終了する
[resetstyle]¥
[endnowait]¥
[s]
```

■インライン画像の表示[graph]

KAGでは、アスキーテキストとして表示可能な文字の他に、オリジナルの絵文字も他の文字と同様に扱うことができます。

これらの絵文字については、前景レイヤーに表示するキャラクターの立ち絵と同様、透明処理されたPNGファイルが推奨されています。

このようなオリジナルの絵文字をKAGでは「インライン画像」と呼び、表示するためには[graph]タグを使います。

●タグ記述例

```
[graph storage="heart.png" char=false]¥
```

上の記述例では、"heart.png"というファイルで提供される絵文字がインライン画像として1文字提供されます。シナリオに書き込む場合は以下のようになります。

●シナリオ例

```
A子：最近[graph storage="heart.png" char=false]してる？[l]
B子：してるよー[graph storage="heart.png" char=false][l]
```

属性「char」に「true」を指定した場合はインライン画像を文字として扱い、「false」を指定すると本来の画像として扱うようになります。

「true」にしておくとインライン画像自体を[font]タグでさまざまな色に変えることができるので、こちらのほうが便利です。

●シナリオ例

```
A子：最近[font color=0xff00ff][graph storage="heart.png"
char=true][resetfont]してる？[l]
B子：してるよー[font color=0xff0000][graph storage="heart.png"
 char=true][resetfont][l]
A子：うお、真っ赤だ。燃えてるね。[l]
B子：うーん、そうでもない。実は[font color=0xff9000][graph
 storage="heart.png" char=true][resetfont]って感じ。[l]
A子：たそがれてるの？[l]
```

このように、１つの「heart.png」をさまざまな色で扱うことができます。

「ff00ff」はピンク色、「ff0000」は真っ赤、ff9000はオレンジ色です。オレンジ色のハートというのも変ですが。

インライン画像の例

■ルビを振る[ruby]

KAGで扱う文字は、それがひらがなだろうがカタカナだろうが漢字だろうが、すべてに対してひらがなだろうがカタカナだろうが漢字だろうがあらゆる種類のルビが振れます。この場合、[ruby]タグを使います。

ワンポイント
だからといって、ひらがなに漢字のルビなど振らないように。

●タグ記述例

```
[ruby text="ふりがな"]
```

[ruby]タグの直後にある文字一文字に対して、text属性で指定した文字がルビとして割り当てられ、横書きの場合は文字の上に、縦書きの場合は右側に表示されます。

ワンポイント
[ruby]タグに関しては次に漢字などの文字がくる都合上、タグ行末に半角の¥は絶対に書きません。

●シナリオ例

```
「[ruby text="う"]盂[ruby text="ら"]蘭[ruby text="ぼん"]盆
[ruby text="え"]会って何でしょうね」[l]
　ぼくが言うと、藤井さんは呆れたように言い返してきた。[l]
「この[ruby text="ご"]期に及んでそんなことはどうだっていいだろう
が。頭を[ruby text="フォ"]初[ruby text="ーマ"]期[ruby text="
ット"]化しろってんだ」[l]
```

ルビのサイズはシナリオ内で任意に変更することはできず、「Config.tjs」内の設定によって決定された値が最後まで引き継がれます。

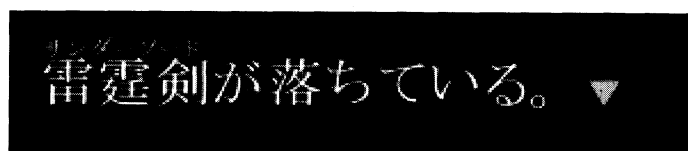
ルビを美しく見せるためには、本文の文字サイズや行間サイズにも注意しなければいけません。

また、これは出版物でも同じことが言えるのですが、1文字や2文字の少

数の文字に長いルビを普通に振るとルビ同士が重なって読めませんし、ルビのサイズを小さくすると、字が見えなくなってしまいます。

このような時は、最初の一文字に対して一気にすべてのルビを振るのですが、そのときにルビテキストの先頭に数文字の空白を入れてルビの位置を手動で調整します。

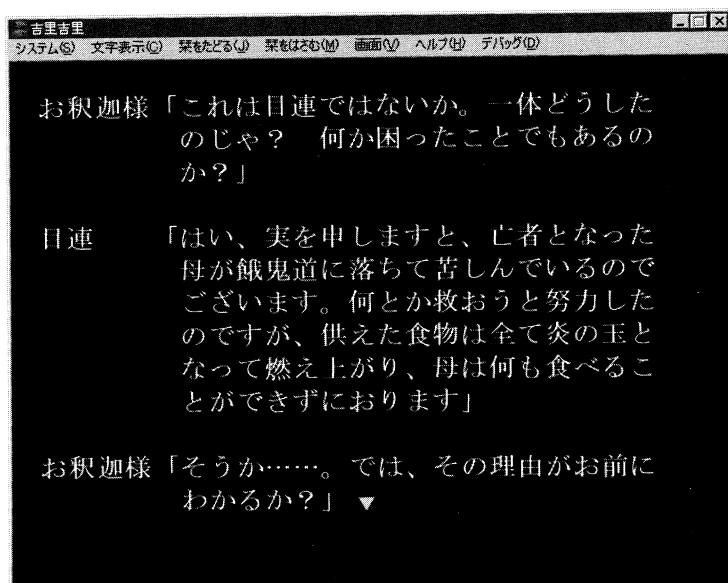
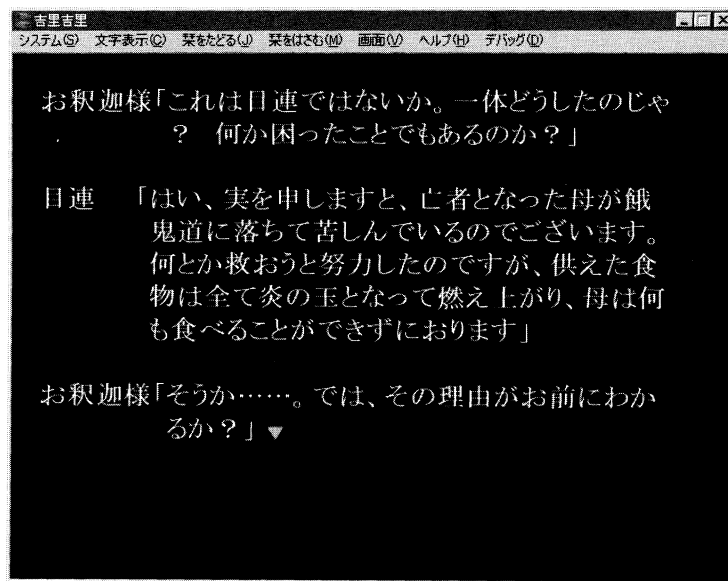
●シナリオ例

```
[ruby text="          サンダーソード"]雷霆剣が落ちている。[l]
```

こうすると、美しいルビを振ることができます。

■インデント機能[indent][endindent]

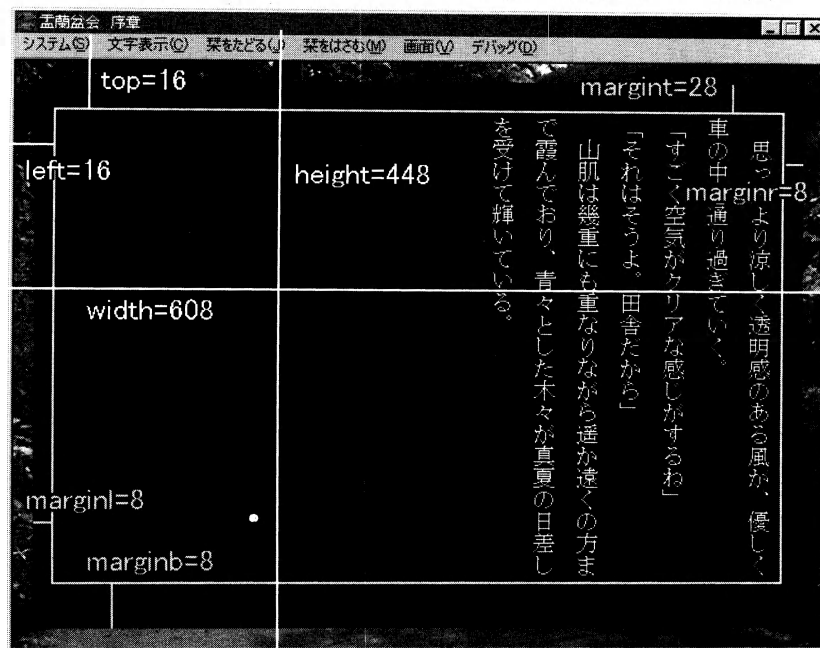
ドラマの脚本のようなレイアウトをしたい場合には、[indent][endindent]タグで提供されているインデント機能を使います。

●シナリオ例

```
[font face="ＭＳ 明朝"]¥
お釈迦様「[indent]これは目連ではないか。一体どうしたのじゃ？　何か
困ったことでもあるのか？」[endindent][l]

目連　　「[indent]はい、実を申しますと、亡者となった母が餓鬼道に落
ちて苦しんでいるのでございます。何とか救おうと努力したのですが、供え
た食物は全て炎の玉となって燃え上がり、母は何も食べることができずにお
ります」[endindent][l]

お釈迦様「[indent]そうか……。では、その理由がお前にわかるか？」
[endindent][l]
```

先頭に[font]タグを置いた例

[font]タグの行を外した例

デフォルトのフォントは「Confit.tjs」内でプロポーショナル・フォントが指定されているので、自動的に字詰めが調整されますが、使いにくい場合はシナリオ例のように等幅フォントを指定するとよいでしょう。

4-4 KAGの文字配置座標について

KAGにおける文字の配置は、以下のような値で定義されています。

●Config.tjs例

```
// ◆ 左右上下マージン
;marginl=8; // 左余白
;margint=28; // 上余白
;marginr=8; // 右余白
;marginb=8; // 下余白
// ◆ 初期位置
;left=16; // 左端位置
;top=16; // 上端位置
;width=608; // 幅
;height=448; // 高さ
// ◆ 右文字マージン
;marginRCh=2;
```

初期位置はメッセージ・レイヤーの黒い四角の位置とサイズを決めており、左右上マージンは、その黒い四角の上に実際に文字が表示される場合の位置を決めています。

禁則処理に使う右文字マージンは、縦書きでは下文字マージンになり、句点や句読点、カッコの終わり、疑問符や感嘆符などが行頭に表示されないようにするためのスペースとして使われます。

これらのすべては[position]タグで変更することができます。

●タグ記述例

```
[position layer=message0 page=fore marginl=8 margint=28
marginr=8 marginb=8 left=16 top=16 width=608 height=448]¥
```

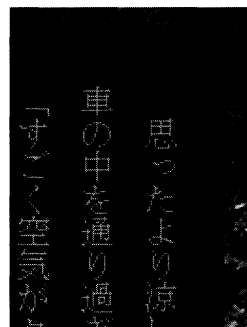
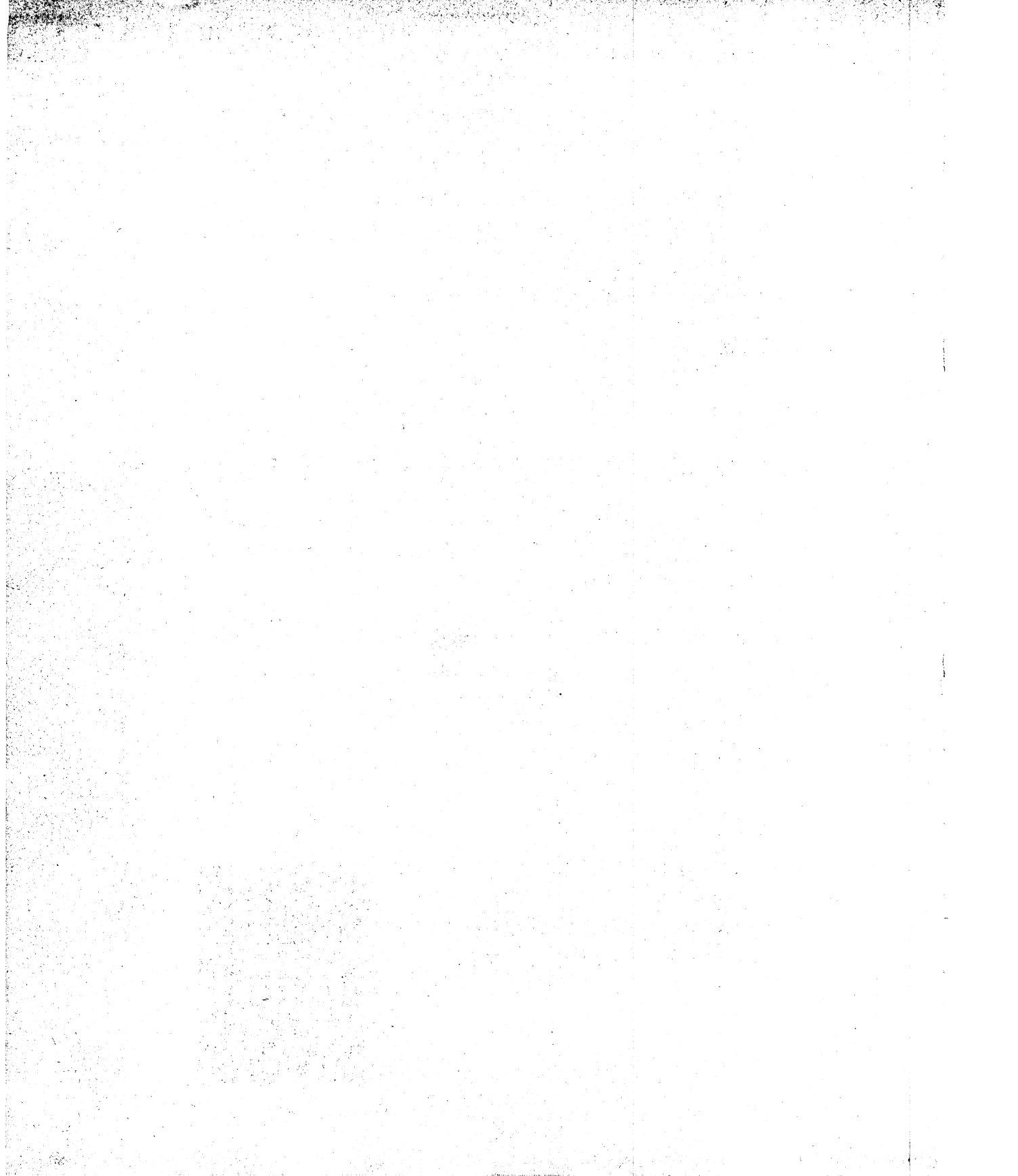
「Congif.tjs」で指定された内容と上記のタグは、同一内容になります。

[position]タグで変更されたメッセージ・レイヤーの属性をリセットするタグは存在しないので、上記のようにデフォルトの値を記述した行を1つ準備しておくと、変更を戻すときに便利です。

文字のサイズや行間については、「Config.tjs」の以下の部分で指定されます。

●Config.tjs例

```
// ◆ 文字の大きさ
;defaultFontSize = 18;
// ◆ 行間
;defaultLineSpacing = 2;
```

「Config.tjs」の行間で設定された幅と、文字の大きさで指定されたフォント・サイズを元に予約された幅が、交互に並びます。文字表示が横書きに設定されている場合は、「幅」ではなく「高さ」になります。

Step 5

選択肢とラベル

分岐処理のあるノベル系ゲームなどを作るには選択肢の知識は必須です。ここでは選択肢文字表示に関するテクニックを解説します。

5-1 選択肢とラベル

市販のアドベンチャーゲームを起動すると、『ゲームを始める』『セーブデータをロードする』『ゲームを終了する』などの選択肢が並んだメニュー画面が表示されますが、これらのメニュー画面はKAGの選択肢機能を利用して作ることができます。

必要なタグは[link][endlink]の二つで、このタグで挟まれた部分がマウスやキーボードで選択可能になり、指定されたジャンプ先にジャンプできるようになります。考え方は、htmlの「<A HREF=***.html>ここをクリックしてね</A>」と同じです。

●タグ記述例

```
[link target=*scene001]右手へ進む[endlink]
[link target=*scene002]左手へ進む[endlink]
[s]
```

target属性には選択されたときにジャンプする先を「*名前」の形式で指定します。これを「ラベル」と呼びます。ラベルは半角の「*」(アスタリスク)印で始まる任意の文字列であることが条件となっており、これが存在しないと、リンクをクリックしたときにエラーになるので注意してください。[link][endlink]タグと半角の「*」印で始まるラベルは切っても切れない関係にあります。一緒に理解するとよいでしょう(ラベルについては次項で詳しく説明します)。

また、[link][endlink]で選択肢を配置した場合は、必ずその最後に[s]タグを置く必要があります。[s]タグはシナリオの実行を強制的に停止するためのもので、プレイヤーが[link][endlink]で配置された選択肢を選ぶのを永遠に待ち続けます。これを置かないと、[link][endlink]の次に書いたシナリオがどんどん実行されてしまい、最悪の場合は画面がクリアされて選択肢が消えてしまう事態が発生してしまいます。

●シナリオ例

```
洞窟は右と左に分かれていた。[l]
どっちへ行こうか。[l]

[link target=*scene01]右手へ進む[endlink]
[link target=*scene02]左手へ進む[endlink]
```

コラム～少[illegible]のひとりごと

htmlを取り扱える人はKAGのタグをそれほど苦もなく理解できると思いますが、逆もまた然り。

KAGのタグ打ちに慣れた方は、自分のホームページをhtmlタグを打ちながら作ってみるのもよろしいかと。

ワンポイント

[link]タグの行は普通のコマンド行ではなく選択肢が表示されるため、行末に半角の¥を記述すると改行されなくなり、すべての選択肢が横一列に並んでしまいます。

選択肢行¥の取り扱いについては注意してください。

ワンポイント

KAGのシナリオは上から書かれた順番に命令を実行するので、[s]タグのような停止タグがない場合はどんどん下に向けて命令を実行し続けてしまいます。

選択肢のようにユーザーにアクションを委ねる場合、[s]タグを置いてシナリオを停止させましょう。

ちなみに、[s]タグの語源はstopの頭文字だとか。

```
[s]
;-----------------------------------------
*scene01
[er]¥
ぼくは右手へ進んだ。[l]
洞窟は行き止まりだった。[l]
ぼくはそこに座り込んだ。[l]
[s]
;-----------------------------------------
*scene02
[er]¥
ぼくは左手へ進んだ。[l]
外への出口が見えてきた。[l]
ぼくは思わず駆け出していた。[l]
[s]
```

上の例における半角の「；」（セミコロン）で始まる行ですが、これはシナリオを見やすくするための「セパレータ」です。シナリオは上の例のようにラベルごとに分けたほうが見やすいので、それぞれの[s]タグの直後に置きます。[s]タグはそこでシナリオの実行を永遠にストップしてしまうので、[s]タグをまたいだ上下のつながりはないためです。

セパレータについては、どんなものでも構いません。半角の「；」（セミコロン）が行頭にある場合、KAGはその行を「注釈文」として解釈し、実行しないからです。

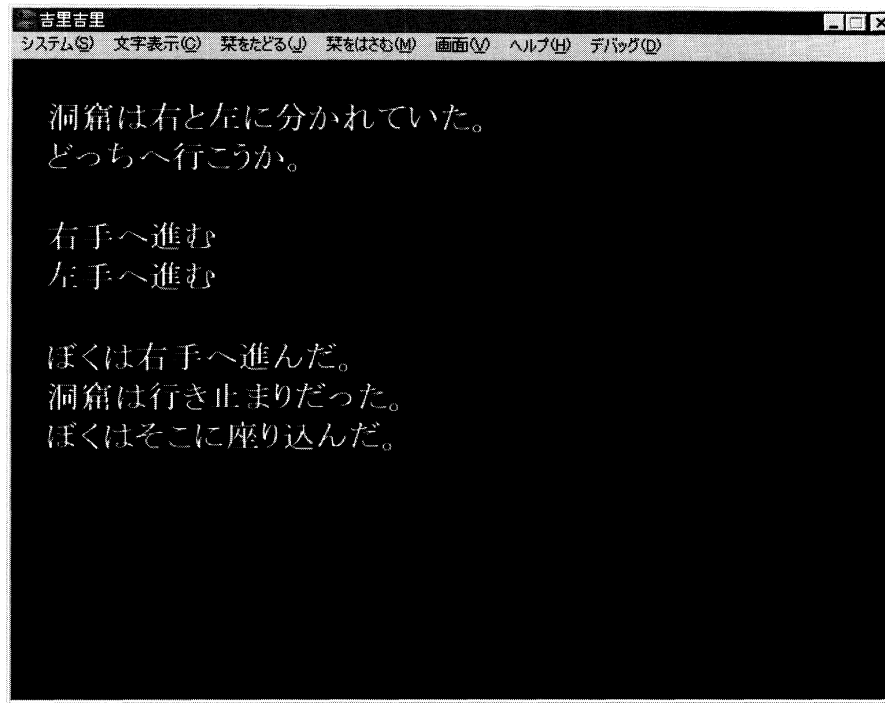
●セパレータ例

```
;○●○●○●○●○●○●○●○●○●○●○●○●

;]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]]

;########################
```

上のセパレータのように、セミコロンの後ろは何でもかまわないので、自分がいちばん見やすいものをセパレータとして決めておくとよいでしょう。

ワンポイント

いくらカッコいいセパレータを作っても、先頭にセミコロン（；）を打ち忘れては意味がありません。このセミコロンはBASICでいう「REM」の意味があるので、シナリオ内に注釈を入れたいときはどんどん打ちましょう。

また、それぞれのラベルの直後に置かれている[er]¥ですが、これがないと、ページが切り替わったように見えず、選択肢の直後にジャンプ先のラベルの内容が表示されてしまいます。

画面表示例

このような状態に陥ることを避けるため、ラベルの直後には必ず[er]や[cm][ct]などの文字消去タグを置いてください。

選択肢を３つ以上使いたい場合も同様で、[link][endlink]で囲まれた選択肢と、ジャンプ先のラベルを増やすだけです。

●シナリオ例

```
*bunki
[er]¥
洞窟は右と左に分かれていた。[l]
「どっちへ行こうか」[l]

[link target=*scene001]右手へ進む[endlink]
[link target=*scene002]左手へ進む[endlink]
[link target=*scene003]その場に座り込む[endlink]
[link target=*scene004]声を上げて泣く[endlink]
[link target=*scene005]今来た道を戻る[endlink]
[s]
;----------------------------------------
*scene001
[er]¥
ぼくは右手へ進んだ。[l]
洞窟は行き止まりだった。[l]
ぼくはそこに座り込んだ。[l]
[s]
;----------------------------------------
*scene002
[er]¥
ぼくは左手へ進んだ。[l]
外への出口が見えてきた。[l]
ぼくは思わず駆け出していた。[l]
[s]
;----------------------------------------
*scene003
[er]¥
ぼくはその場に座り込んだ。[l]
もうどうなってもいいや。[l]
[s]
;----------------------------------------
*scene004
```

```
[er]¥
ぼくは声を上げて泣いた。[l]
涙はしょっぱかった。[l]
[s]
;--------------------------------------
*scene005
[er]¥
ぼくは今来た道を戻ることにした。[l]
しかし、背後からはあの怨霊が追いかけてくるはずだ。[l]
やっぱり戻れない！[l]
[jump target=*bunki]¥
[s]
```

上の例では、「選択肢５」を選んだ場合のみ、もう一度選択肢を選び直させるために[jump]というタグを使っています。この[jump]タグは、[link][endlink]とは違って、プレイヤーが選択しなくても自動的に処理を指定のラベルに移す働きをもっています。

[jump]タグについてですが、BASICをご存知の方はgotoコマンドを思い出してください。あれとまったく一緒です。

●タグ記述例

```
[jump target=*どこかのラベル]¥
```

[jump]タグは強制的に処理をジャンプさせるので、その下に書かれているシナリオは[jump]より上に書かれたシナリオとは無関係になります。このため、慣れないうちは[jump]の直後に[s]を書いておくと「ああ、ここで処理の流れが止まっているんだな」と理解しやすくなります。

5-2 ラベルとセーブ・ポイント

吉里吉里／KAGで制作したゲームにはメニューに「栞をたどる」「栞をはさむ」という２つのコマンドがありますが、これは一般に言われる「ロード」「セーブ」のことです。

KAGで作られた作品でプレイ状況をセーブする場合、ユーザーは自由な場所でセーブすることはできません。セーブ可能な場所はKAGシナリオ上においてラベルが書かれている行のみです。

さらに注意なのは、ラベル行にセーブ・タイトルが記述されていないとセーブできないことです。

●セーブできないラベルの例

```
*start
*opening
*scene01
```

●セーブできるラベルの例

```
*start|スタート
*opening|オープニング
*scene01|出会い
```

ラベルの後ろに半角の縦棒（|）を置き、その後ろにセーブ・タイトルを書いておけば、プレイヤーはそのラベルでセーブできるようになります。

セーブした場合、セーブ・タイトルは栞に表示されます。上の例では、ラベル「*opening」でセーブしたら栞には『オープニング』と表示され、ラベル「*scene01」でセーブしたら栞には『出会い』と表示されます。

選択肢を置く場合は必ずラベルを書かねばなりませんが、この場合、セーブ・タイトルは必須ではありません。そこでセーブさせたくないと思った場合は、セーブ・タイトルを省略して純粋なラベルだけ書いておけばよいのです。

この場合、プレイヤーが「栞を挟む」を使った場合、そこから一番近い過去に存在するセーブ・タイトルつきのラベルを通過した時の状態が、セーブデータとして栞に保存されます。

次の例を見てみましょう。

ワンポイント

半角の縦棒（|）はキーボードではBackSpaceキーの左にあります。この記号は正式には「パイプライン」と呼びます。

ワンポイント

セーブ・タイトルがあまり長いと、メニューが幅広になってしまって使いにくくなります。

セーブ・タイトルには、短くて意味がよく分かって、プレイヤーのゲーム魂を刺激するような粋な言葉を付けられたらいいですね。

●タグ記述例

```
[title name="1999ChristmasEve"]¥
*start|オープニング
[er]¥
　ぼくは立花明。どこにでもいる普通の社会人。[l]
　今日は12月24日。世間一般で言う、クリスマスイブ。[l]
　イブの夜と聞いて、まず思いつくのは……[l]
　ベイエリアの高層シティホテル。よく冷えたシャンパン。窓辺に飾った薔
薇の花束。そして小さな箱に入ったクリスマスプレゼント。[l]
　もしくは、小さなペンションホテル。お揃いのロシニョールのスキー板。
暗い窓の外に舞う粉雪。そしてテーブルの上のキャンドルライト。[l]
　え？　その後……？[l]
　その後って……雑誌にはそれしか書いてなかったけど……？[p]
[er]¥
　まあ、それはさておき、昨年のクリスマス。[l]
　ぼくは、数年前からひそかに恋していた女性を豪華なディナーに誘うこと
にしていた。[l]
　女性の名は桐野由美香。[l]
　勤め先は別だが、ぼくと同い年で、ちょっとしたきっかけで知り合ってか
らはまるで兄弟のように仲のいい友達だ。[l]
　友達という間柄に妙な恋愛感情が混入するとつきあい方がギクシャクして
くる前例は数多いが、ぼく達の場合はそうではない。[l]
　ぼく自身は確かに由美香のことは好きだ。でも、それを簡単に口にできる
ような男でもないし、言ったからどうなるとも思っていないのだ。[l]
「愛してるよ」……[l]なんて、逆立ちしても無理だ。[p]
[er]¥
　そんなこんなで、店の予約をしておきながらもぼくはなかなか本人を誘う
ことができなかった。[l]
　そして結論から言えば、ぼくはディナーの予約を恋人のいる友達に売りつ
けてしまった。[l]
　ぼくがやっと勇気を奮い起こして告げた時は、容姿端麗、明眸皓歯、才色
兼備の由美香にはすでに先約が入っていたのだ。[l]
　由美香はこんなふうに言っていた。[l]
「んもう、言い出すのが遅いわよ明。今年は友達とスキーに行くの。でも、
```

```
もし明さえよければ、来年のイブは今から空けておくわ」[l]
　何とも気の長い話に聞こえるが、由美香の言葉としては別におかしくない。
単純なぼくは、早速、一年先の計画を立て始めた。[p]
[er]¥
　そしていよいよ、今年のクリスマスシーズン。[l]
　旅行先については、昨年の冬に発売されたパソコンゲーム『かまいたちの
聖夜』にあやかって長野県白馬村のペンションホテルに決めた。[l]
　実はこのゲームにはいわくがある。[l]
```

この例では、途中に[er]が3つ書かれていることから、文章は全部で4ページに渡って表示されます。

たとえば、これをプレイしていて、最後の行の『実はこのゲームにはいわくがある』という文章を読んだところで栞を挟み、その栞を呼び出した場合、どこからゲームが再開されるかというと、1ページ目の冒頭『ぼくは立花明。どこにでもいる……』から始まります。

つまり、KAGの栞はいちばん最近に通過したラベルの位置を覚えていて、その時点でのデータをセーブするのです。

それぞれのページ頭でセーブしたい場合は、[er]¥の行の直前に以下のようなラベルを記述します。

●タグ記述例

```
*start|オープニング
[er]¥
　ぼくは立花明。どこにでもいる普通の社会人。[l]

*start2|オープニング
[er]¥
　まあ、それはさておき、昨年のクリスマス。[l]

*start3|オープニング
[er]¥
　そして結論から言えば、ぼくはディナーの予約を恋人のいる友達に売りつ
けてしまった。[l]
```

```
*start4|オープニング
[er]¥
　そしていよいよ、今年のクリスマスシーズン。[l]
```

このようにすべてのページにラベルを記述していくのは作業的に厳しいのですが、これ以外に、各ページでセーブさせる方法はありません。

また、上記の例ではセーブ・タイトルがすべて『オープニング』になっていますが、セーブ・タイトルはラベルと違って重複可能です。もし、同じセーブ・タイトルを記述するのが面倒な場合は、下記のように書きます。

●タグ記述例

```
*start|オープニング
*start2|
*start3|
*start4|
```

ラベルの右側に「|」だけ書けば、その直前に通過したセーブ・タイトルが引き継がれます。直前が同様に「|」だけの場合は、さらに遡っていちばん近いセーブ・タイトルを保存します。

また、ラベルの直後の行には[er][cm][ct]などの文字消去タグを必ず置くようにします。この理由は、KAGが栞に保存されたセーブ・データをロードするときに、必ず画面クリアを行なって、そこに書かれている文字をすべて消去してしまうからです。

KAGでは、画像レイヤーの状態、ロードされているファイルの情報、サウンド・バッファやサウンド・ファイルの情報などは栞に保存しますが、メッセージ・レイヤーのどの位置にどの文字が表示されていたかという情報までは保存しません。

このため栞をロードしたときには常に画面がクリアされるわけですから、その挙動と、普通に読み進めているときの状態を等しくしておく必要があり、このためにラベルの直後ではメッセージ・レイヤーをリセットする[er][ct][cm]タグを書く必要があるのです。

ワンポイント

長編の場合は、特に必要でない場所（たとえば長いシーンの中ほど）にもラベルを書くとよいかもしれません。

10ページも20ページもラベルのないシナリオを書いてしまうと、セーブ・ポイントがはるか過去ということになり、プレイヤーはロードするたびに苦痛を感じるはずです。

選択肢が途中にある場合は否応なくラベルが存在しますが、そうでない場合は3～5ページごとに適宜セーブ・ポイントを作ってあげたほうがいいでしょう。

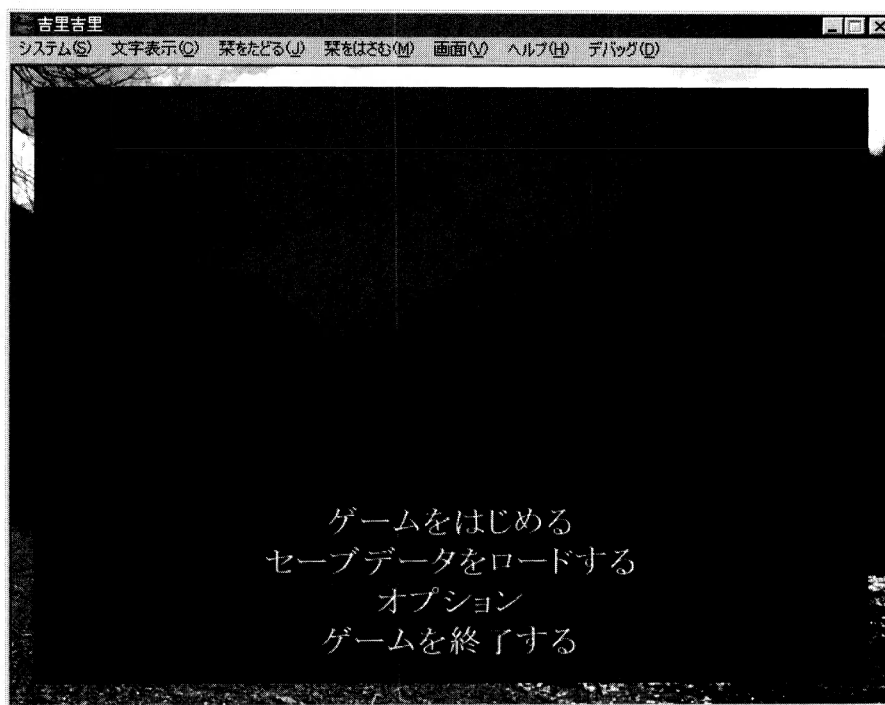
■メニューの作成例

一例ですが、メニューを作るには以下のように書きます。

●シナリオ例

```
*start
[position layer=message0 page=back frame="" opacity=200]¥
[playbgm storage="test.mid"]¥
[image storage="test01.bmp" page=back layer=base]¥
[current layer=message0 page=back]¥
[locate y=260]¥
[nowait]¥
[style align=center]¥
[link target=*opening]　　ゲームをはじめる　　[endlink]
[link target=*dataload]セーブデータをロードする[endlink]
[link target=*option]　　　オプション　　　[endlink]
[link target=*end]　　ゲームを終了する　　[endlink]¥
[style align=default]¥
[endnowait]¥
[trans method=crossfade time=2000]¥
[wt]¥
[current layer=message0 page=fore]¥
[s]
;--------------------------------------------
*opening
[cm]¥
オープニング。[l]
[s]
;--------------------------------------------
*dataload
[cm]¥
データロード。[l]
[s]
;--------------------------------------------
*option
```

```
[cm]¥
オプション。[l]
[s]
;-------------------------------------------
*end
[cm]¥
エンド。[l]
[s]
```

[position]タグは、以前メッセージ・レイヤーの位置やサイズを変更するときに説明しましたが、ここでは画像枠や透明度などの属性を指定するために利用します。

画面例

●タグ記述例

```
[position layer=message0 page=fore frame="waku.png"
opacity=255]¥
```

frame属性は、メッセージ・レイヤーに画像枠を使いたい場合にファイル名を指定する属性です。画像枠が不要な場合は「frame=""」のように記述します。下の画面例では、画面下部に「メッセージ・レイヤー0」がメッセージ枠と一緒に表示されています。

メッセージ枠を使った画面例

opacity属性は、メッセージ・レイヤーの濃度を指定する属性で、「255」のときは完全な可視状態で、「0」にすると完全に透明になります。シナリオの画面例では「opacity=200」ですが、上の画像枠を使った画面例では「100」にしてあります。

シナリオ例の5、17行目にある[current]タグは初めてだと思いますが、これは、文字を表記するためのメッセージ・レイヤーや、その裏表を指定するためのタグです。

●タグ記述例

```
[current layer=message0 page=back]¥
[current layer=message0 page=fore]¥
[current layer=message1 page=fore]¥
```

ここでは、メッセージ・レイヤー0のバックグラウンドに文字を表示し、背景画像と一緒にトランジションで表へ表示するために利用しています。トランジションが終了したら、きちんとメッセージ・レイヤー0のフォアグラウンドに戻しておきます。

選択肢として存在する4つの[link][endlink]部分に全角のスペースが記述されていますが、これはマウス・カーソルが乗ったりカーソル・キーで選択されたときに表示される矩形の大きさを揃えるためです。ただし、この方法は、デフォルト・フォントがプロポーショナル*ではなく等幅フォント*の場合のみ利用できます。

また、[link][endlink]の最後の行の行末に半角の「¥」が置かれているのは、もしこの「¥」がないと、改行扱いになってしまって、次のページに進むためのクリック待ち記号が表示されてしまうからです。これは文字列が画面の下端に到達したことが原因ですが、メニューが改ページしてしまうのは避けなければならないので、最後の行に改行を中止する「¥」を置くことで回避しています。

*プロポーショナル・フォント
proportional font
文字幅がアルファベットによって異なるフォント。アルファベットはそれぞれ文字幅が異なるため、きれいな文字組みを表現するために使われる。

*等幅フォント
文字の形に関係なく、全角の半分の幅をもつフォント。

Step 6
変数の利用
ただ読ませるだけの電子小説を作ることに飽きたら、パラメータによる分岐処理、乱数、フラグによるイベント管理など、変数を利用してプレイヤーがあっと驚く作品を作ってみましょう。ここでは変数の基本的な取り扱いについて解説します。

6-1 KAGで取り扱える変数の形式

ワンポイント
変数は「何かを入れておく入れ物」のことです。
中学の数学で学ぶxやaなどがこれに当たります。

ゲームの主人公の名前などをプレイヤーに入力してもらうためには、変数を利用します。

KAGでは、「数値変数」「文字変数」「フラグ」に区別はなく、すべて同一のものとして扱うことができます。このために変数に数値を代入すれば「数値変数」として、文字を代入すれば「文字変数」として、「0」か「1」を代入すれば「フラグ」として利用することができます。

≪変数の名前のルール≫

変数を使いたい場合、まずは変数に名前をつけますが、このとき、以下のようなルールを守ります。

- 半角英数と全角文字のみ利用可
- 記号は「_」(アンダーバー) のみ利用可
- ゲーム変数の名前は「f.」、システム変数の名前は「sf.」で始める
- 変数名の先頭に数字を使ってはいけない
- 予約語は使ってはいけない

●予約語の例

```
count assign save load finalize invalidate instanceof
continue function debugger property default extends
finally isvalid typeof delete return export import switch
global setter getter break false super catch class while
throw const enum this void true null with else case for
new var try in if do
```

これらの単語はTJS内で予約されているため、変数として使うことができません。たとえば、「f.count」や「sf.assign」という変数名は使えないので、注意してください。

≪ゲーム変数とシステム変数の違い≫

「ゲーム変数」はゲーム・シナリオ内での処理に利用し、「システム変数」はゲームのシナリオを離れたシナリオの外で利用します。

ゲーム変数は個々のセーブ・データに記録されるため、「吉里吉里」を再起動した場合は読み込まれません。このため「見たエンディング／見ていないエンディング」などを表わす変数データは保持することができません。

システム変数は「吉里吉里」を再起動したときに真っ先に読み込まれるファイル内部に記録されているため、どのセーブ・データをロードしても、最初からプレイをやり直しても、最後に代入された値が常に保持されます。

ゲーム内の処理をするときに「ゲーム変数」を用い、シナリオ外のシステム的な処理に「システム変数」を使うことを覚えておいてください。

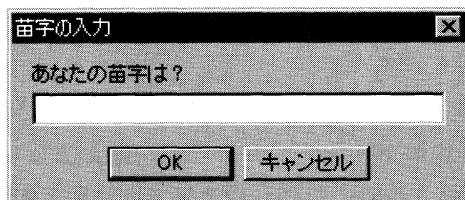
ワンポイント

変数に名前をつける場合は日本語をローマ字で表記した名前をつけると、予約語とぶつからずに済みます。
ローマ字ではかっこ悪いと思う方は、予約語リストを覚えて意図的に回避してください。

●変数例

```
f.okane  f.hitpoint  f.magicpoint  f.koukando  f.pastime
sf.ending01  sf.name  sf.pictureshow  sf.wdee など
```

■変数を使った名前入力

ここでは、「あなたの名前を入力してください」と促すダイアログを開き、入力してもらった文字列を変数に格納して表示してみます。

名前を入れておくための文字変数は「f.name」にしたいところですが、苗字と名前を別々の変数に格納したいので、「f.sname」「f.name」と2つ準備します。

続いて、画面に入力ウインドウを表示してプレイヤーに名前を入力させますが、ここで利用するのは[input]タグです。

●タグ記述例

```
[input name=f.sname prompt="あなたの苗字は？" title="苗字の入力"]¥
[input name=f.name prompt="あなたの名前は？" title="名前の入力"]¥
```

「name属性」は入力文字列や数値を格納するための変数名で、「prompt属性」は入力ダイアログの内部に表示されるメッセージ、「title属性」はダイアログの左上のタイトルに表示するメッセージをそれぞれ指定します。

また、変数の内容をシナリオ内で表示するには、[emb] タグを使います。

●タグ記述例

```
[emb exp=f.sname1]
```

●シナリオ例

```
*inputname
[er]¥
あなたの名前を教えてください。[l]
[input name=f.sname prompt="あなたの苗字は？" title="苗字の入力"]¥
[input name=f.name prompt="あなたの名前は？" title="名前の入力"]¥
あなたの名前は[font color=0xffff00][emb exp=f.sname][emb
exp=f.name][resetfont]でいいですか？
[link target=*ok]いい[endlink]
[link target=*inputname]だめ[endlink]¥
[s]
```

上の例では、何も文字が入力されていない状態でエンター・キーやOKボタンを押した場合、変数が空の状態でシナリオが進んでしまいます。このため、入力ダイアログを表示する前に最初からサンプルの文字列を変数に代入しておけば、名前が空白のままでシナリオが始まるというトラブルが起きません。

変数に数値や文字などの値を恣意的に代入するには[eval]タグを使います。

●タグ記述例

```
[eval exp="f.sname='ＰＩＡ'"]¥
[eval exp="f.name='少尉'"]¥
```

これで「f.sname」には「ＰＩＡ」という文字列が、「f.name」には「少尉」という文字列が入りました。

KAGシナリオ内で文字列を指定する場合は必ずシングル・クオーテーション（'）で文字列をくくる癖をつけてください。

■変数内容の評価による条件分岐

長い名前や、使ってほしくない名前を使われた場合の他に、何も入力しなかったり、入力されていたサンプルを消してしまった場合、これらの変数内容を評価し、必要なら再入力させることができます。

これらは条件分岐なので、[if]タグを用います。[if]タグは必ず[endif]とペアで使います。

まずは変数 f.nameに入力された名前が『少尉』のときは入力をやり直させるシナリオ例です。

●シナリオ例

```
*inputname
[er]¥
あなたの名前を教えてください。[l]
[input name=f.name prompt="あなたの名前は？" title="名前の入力"]¥
[if exp="f.name=='少尉'"]¥
[jump target=*inputname_02]¥
[endif]¥
あなたの名前は[font color=0xffff00][emb exp=f.name][resetfont]
でいいですか？
[link target=*ok]いい[endlink]
[link target=*inputname]だめ[endlink]¥
[s]
```

この場合、「f.name」という名前の変数に「少尉」という文字列が入力されている場合は[if][endif]の間が実行されます。つまり[jump]タグによるラベルジャンプが行なわれるわけです。

次に、変数「f.name」の内容が空の場合に入力をやり直させるシナリオ例です。

ワンポイント

[if]による条件分岐において前後が等しい場合はイコールは２つ必要です。これを１つしか書かないと意図した処理ができません。このときエラーは返らないので、注意してください。

条件分岐の詳細については「数値変数」の項で詳しく説明しています。

●シナリオ例

```
*inputname
[er]¥
あなたの名前を教えてください。[l]
[input name=f.name prompt="あなたの名前は？" title="名前の入力"]¥
[if exp="f.name==''"]¥
[jump target=*inputname_02]¥
[endif]¥
あなたの名前は[font color=0xffff00][emb exp=f.name][resetfont]
でいいですか？
[link target=*ok]いい[endlink]
[link target=*inputname_02]だめ[endlink]¥
[s]
```

シングル・クオーテーションを2つ並べて記述すると、「空」の意味になります。また、これらの2つの条件分岐を1つにまとめることもできます。

●シナリオ例

```
*inputname
[er]¥
あなたの名前を教えてください。[l]
[input name=f.name prompt="あなたの名前は？" title="名前の入力"]¥
[if exp="f.name=='少尉'||f.name==''"]¥
[jump target=*inputname_02]¥
[endif]¥
あなたの名前は[font color=0xffff00][emb exp=f.name][resetfont]
でいいですか？
[link target=*ok]いい[endlink]
[link target=*inputname_02]だめ[endlink]¥
[s]
```

半角の「｜」を2つ続けて記述すると、「OR」の意味になり、どちらかの条件を満たしていれば「exp」による評価式が“真”になります。3つ以上の条件を処理したい場合も、同様に評価式を延長して記述することによって処理できます。

ワンポイント

ORに利用するのは「||」で、ANDには「&&」を利用します。使い方は「||」と同じです。

●記述例

```
[if exp="f.name=='少尉'||f.name==''||f.name'Dee'||f.name
'でぃー'"]¥
```

長すぎる名前を受け付けなくするには、文字変数に入力された文字列が何文字かを調べ、それを評価する式を作って入力制限を行ないます。

変数に格納されている文字の長さを調べるには、変数の直後に「.length」と書きます。

●記述例

```
f.name.length
```

●シナリオ例

```
*inputname
[er]¥
あなたの名前を教えてください。[l]
[input name=f.name prompt="あなたの名前は？" title="名前の入力"]¥
[if exp="f.name.length>3||f.name==''"]¥
名前は１から３文字までです。[l]
再入力してください。[l]
[jump target=*inputname_03]¥
[endif]¥
あなたの名前は[font color=0xffff00][emb exp=f.name][resetfont]
でいいですか？
[link target=*ok]いい[endlink]
[link target=*inputname_03]だめ[endlink]¥
[s]
```

入力された文字が、ある特定のキーワードを含んでいるかどうかを調べる場合は、「.indexOf」を用います。

「indexOf」は、「A.indexOf('B')」の記述形式をとり、文字列Aの中で文字列Bが現われる位置を調べます。文字列Aの先頭に文字列Bがあれば返り値は「0」になり、文字列Aの2文字目に文字列Bがあれば返り値は「1」になります。

文字列Aの中に文字列Bが見付からない場合は返り値は「-1」になるので、これを評価することで、文字列Aの中に文字列Bが含まれているかを調べることができます。

●記述例

```
*inputname
[er]¥
あなたの名前を教えてください。[l]
[input name=f.name prompt="あなたの名前は？" title="名前の入力"]¥
[if exp="f.name.indexOf('ばか')!=-1"]¥
悪口は言わないで、ちゃんとした名前を入力してください。[l]
[jump target=*inputname_04]¥
[endif]¥
あなたの名前は[font color=0xffff00][emb exp=f.name][resetfont]
でいいですか？
[link target=*ok]いい[endlink]
[link target=*inputname_04]だめ[endlink]¥
[s]
```

上記の例では、「f.name」という変数に「ばか」という文字列があるかどうかを調べています。もし「f.name」の中に「ばかじゃん」や「大ばか」、「かばかば」など、「ばか」という文字列が使われていた場合は「indexOf」への返り値は「-1」以外になり、[if]タグで定義された条件を満たすので、2行目のメッセージが表示されます。

6-2 数値変数

■数値変数のフラグ利用

続いて数値変数の利用ですが、数値変数は大きく分けて「数値変数」と「フラグ」の2種類の使い方ができます。

まずは「フラグ」ですが、市販のゲームではプレイヤーが何かのアクションを行なった場合にスイッチをONにし、後の条件分岐の時にスイッチの状態によって分岐先を決定することが多々あります。このスイッチのことを「フラグ」と呼びます。

●シナリオ例

```
*scene01|旅人
[ct]¥
　旅人が食料をめぐんでくれと言っています。どうしますか？[l]

[link target=*scene02]めぐんでやる[endlink]
[link target=*scene03]めぐんでやらない[endlink]
[s]
;------------------------------------------------
*scene02|めぐむ
[ct]¥
　ぼくは旅人に食料を少し分けてやった。[l]
「あんたは優しい人だな。よし、一ついいことを教えてやろう」[l]
　旅人はぼくが渡したパンを食べながら話し出した。
「あの洞窟の床には、一ヶ所だけ石床が外れる場所がある。その下に宝が埋まっているって話だ。ま、俺は怪物に食い殺されるのは嫌だから行かないが、腕に自身があるなら行ってみたらどうだい？」[l]
[eval exp="f.hint=1"]¥
[jump target=*scene04]¥
[s]
;------------------------------------------------
*scene03|めぐまない
[ct]¥
　ぼくは旅人を無視することにした。[l]
```

```
 これから洞窟へ入らねばならないのだから。[l]
[eval exp="f.hint=0"]¥
[jump target=*scene04]¥
[s]
;-----------------------------------------------
*scene04|洞窟
[ct]¥
 ぼくは洞窟へ入っていった。[l]
[if exp="f.hint==1"][jump target=*scene04b][endif]¥
 だが、洞窟には何もなかった。[p]
 [s]
;-----------------------------------------------
*scene04b
 しばらく歩いたところで、ぼくは先ほどの旅人の話を思い出し、床石を調
べてみることにした。[l]
 なるほど、確かに一ヶ所だけ外れそうな床石がある。[l]
 [s]
```

旅人に食料をめぐむと、「f.hint」という変数の値が「1」になります。これは、「旅人に洞窟の宝の話を聞いた」というフラグがONになったことを意味します。これらを利用すれば、後に[if]タグで条件分岐を行なうときに、プレイヤーのプレイ状態に従ってシナリオを分岐できるわけです。

また、選択肢を選んだ時点でフラグの状態を変化させることもできます。この場合は、[link]タグにexp属性を記述します。

●シナリオ例

```
*scene01|旅人
[ct]¥
 旅人が食料をめぐんでくれと言っています。[l]
 どうしますか？[l]
[link target=*scene02 exp="f.hint=1"]めぐんでやる[endlink]¥
[link target=*scene03 exp="f.hint=0"]めぐんでやらない
[endlink]¥
[s]
```

KAGの場合、変数を[eval]で指定しなければ、その変数は未定義の状態で扱われます。つまり、「0」の値を代入したい場合は、以下の記述例のように何も書かなくてもかまわないのです。

●シナリオ例

```
*scene01|旅人
[ct]¥
 旅人が食料をめぐんでくれと言っています。[l]
 どうしますか？[l]
[link target=*scene02 exp="f.hint=1"]めぐんでやる[endlink]¥
[link target=*scene03"]めぐんでやらない[endlink]¥
[s]
```

このように変数に値が入る方のみ記述しておけば、後に[if]タグで条件分岐させた場合に、「f.button」が未定義の場合は「値＝０」として利用できるわけです。

■数値変数の演算と条件分岐

KAGで一度定義した変数の内容は、シナリオ内で意図的に消去することがない限り、シナリオ実行中はずっと保持されます。

たとえば、主人公のライフ・ポイントのように、ゲーム・スタート時には「10」で、罠にはまると減り、薬を飲むと回復するような値を管理するために使えます。

●タグ記述例

初期設定	[eval exp="f.lp=10"]¥
現在の値に３を追加	[eval exp="f.lp=f.lp+3"]¥
現在の値から３を減	[eval exp="f.lp=f.lp-3"]¥
現在の値の３倍	[eval exp="f.lp=f.lp*3"]¥
現在の値の３分の１	[eval exp="f.lp=~~(f.lp/3)"]¥

初期設定だけは「f.lp」に「10」という整数を代入していますが、それ以降はすべてイコールの右にも左にも「f.lp」が記述されていて、ちょっと変な式に見えますよね。これは、数学的に考えるとおかしいのですが「イコールの右側の値をイコールの左側の変数に代入する」という意味で、プログラムでは多く利用されている表記方法なのです。

ですから[eval exp="f.lp=f.lp+3"]という式は、「現在の“f.lp”（右側）に“3”をプラスして、それを新しい“f.lp”（左側）の値として保管しなさい」という意味になります。割り算の部分だけは「~~」（チルダ）とカッコが書かれていますが、これは割り算の値が整数以外になってしまった場合に、商を切り捨てて整数化するためのものです。また、割り算で商ではなく余りを得たい場合は、半角スラッシュではなく、以下のように半角の「%」を使います。

●タグ記述例

```
[eval exp="f.余り=f.割られる数%f.割る数"]¥
```

●シナリオ例

```
*start
[er]¥
[eval exp="f.apple=10"]¥
りんごが10個あります。[l]
何人で分けますか？[l]
[input name=f.nannin prompt="何人で分ける？" title="人数の入力"]¥
[emb exp="f.nannin"]人で分けました。

[eval exp="f.hitori=~~(f.apple/f.nannin)"]¥
一人あたり[emb exp="f.hitori"]こです。

[eval exp="f.amari=f.apple%f.nannin"]¥
余りは[emb exp="f.amari"]こです。

[link target=*start]もう一度やる[endlink]
[s]
```

●変数の覚書

変数	内容
f.apple	りんごの総数
f.nannin	分ける人数
f.hitori	一人あたりのりんごの数
f.amari	あまり

変数を複数使った処理を行なう場合は、このように変数の覚書を作っておくとよいでしょう。

続いて、変数の内容による条件分岐ですが、以下の例のように記述することによって[if]タグでその後の処理を分岐させることができます。

まずは変数「f.okane」の値が500以下の場合のみお金を500もらえる条件分岐のシナリオ例です。

●シナリオ例

```
[if exp="f.okane<=500"]¥
[eval exp="f.okane=f.okane+500"]¥
[endif]¥
```

続いて、変数「f.okane」の値が500以下の場合のみお金を500もらい、合計金額が10000を超えた場合は全部奪われてしまう条件分岐のシナリオ例です。

●シナリオ例

```
[if exp="f.okane<=500"]¥
[eval exp="f.okane=f.okane+500"]¥
[if exp="f.okane>10000]¥
[eval exp="f.okane=0"]¥
[endif]¥
[endif]¥
```

最後に、変数「f.okane」の値が500以下の場合のみ、現在のお金を2倍にしてもらい、さらに1000もらえるというシナリオ例です。

●シナリオ例

```
[if exp="f.okane<=500"]¥
[eval exp="f.okane=f.okane*2+1000"]¥
[endif]¥
```

[if] タグの復習になりますが、このタグのexp属性で行なっていることは、「変数と文字」、または「変数と数値」、または「変数と変数」の照合です。もしこの照合結果が『一致』(つまり真)である場合は、[if] タグと [endif] タグの間にあるタグが実行されたり、文章が表示されたりし、照合結果が『不一致』(つまり偽)の場合は、[if] タグと [endif] タグの間の行は無視されます。

[if]の記述をまとめると、下のようになります。

```
[if exp="変数や定数　等号不等号　変数や定数"]¥
(条件に一致している場合の処理)
[endif]¥
```

上記の例は下のように1行でも書けます。

```
[if exp="変数や定数　等号不等号　変数や定数"] (条件に一致している場合の処理) [endif]¥
```

変数や定数を書く場合ですが、数値や変数の場合はそのまま記述できます。

●記述例

```
[if exp="f.okane==1000"]
```

文字列の場合は、ダブル・クオーテーション (") でくくります。

●記述例

```
[if exp="f.hannin=='キヨ'"]
```

また、左右の定数や変数、文字列などを比較するための等号不等号については、以下の種類があります。

==	前と後ろが等しい
>=	前が後ろ以上
>	前が後ろより大きい
<=	前が後ろ以下
<	前が後ろより小さい
!=	前と後ろが異なる

ワンポイント

比較演算子の前後が等しい場合はイコールは2つ必要です。私の体験から言っても、他の部分でイコールを1つしか書かないので、つい間違えて記述してしまいがちです。[if]タグが動作していないと思った場合は、まずは等価の場合のイコールが2つ書かれているかどうかを確認するとよいでしょう。

6-3 乱数の利用

「じゃんけんゲーム」や「ルーレット」などは、こちらが意図した結果をコンピュータが出してこない点が面白いのですが、これらを実現するためには、「乱数*」を使います。

KAGでは、乱数を発生させるジェネレータとして「random」と「intrandom」というものが準備されており、「random」は0以上1未満の実数*を、「intrandom」は指定値以上、指定値以下の整数*の乱数を生成します。KAGを利用してゲームを作る場合は、「intrandom」の方を利用します。

*乱数
でたらめな数値

*実数
誤差を含む数。小数点のついた数。

*整数
0と自然数、逆自然数。小数点の無い数。

●記述例

```
[eval exp="f.ransuu=intrandom(1,100)"]¥
```

上の例では「f.ransuu」という変数に「1」から「100」のいずれかが代入されます。

●シナリオ例

```
*start
[er]¥
さいころ勝負！[l]
[eval exp="f.dice_player=intrandom(1,6)"]¥
[eval exp="f.dice_com=intrandom(1,6)"]¥
コンピューターの目：[emb exp="f.dice_com"]
プレイヤーの目　　：[emb exp="f.dice_player"]
[if exp="f.dice_com>f.dice_player"]コンピュータの勝ち！
[r][endif]¥
[if exp="f.dice_com<f.dice_player"]プレイヤーの勝ち！
[r][endif]¥
[if exp="f.dice_com==f.dice_player"]引き分け！[r][endif]¥
[link target=*start]もう一度やる[endlink]¥
[s]
```

さいころの目は1から6までなので「intrandom(1,6)」という記述を行ない、それを[eval]タグを用いて「f.dice_player」と「f.dice_com」という変数にそれぞれ代入しています。あとはそれらを[emb]タグで表示し、[if]で双方の値を比較してメッセージを表示します。

6-4 数値変数処理部のサブルーチン化

ゲームを作っていると、定期的に変数の処理を行なわねばならないような場面に出くわす場合があります。

例として、拙作「1999ChristmasEve」の第二話に厳冬の森の中を歩くシーンがありますが、ここでは別のラベルに進むごとに時間が5分経過し、どこをどう歩いても120分を過ぎると自動的に凍死するシーンへジャンプする処理をしています。

シーンは全部で50近くもあるので、各ラベルの冒頭に数値変数の処理をするためのシナリオを書いていたのでは後で修正するときにも面倒ですし、何よりシナリオが汚くなってしまうので、サブルーチンを使っています。

「サブルーチン*」は、定型処理をするシナリオ部を一つだけ作っておき、必要に応じてそこにアクセスして処理を行なうことができるもので、これを呼び出すには[call]タグを、戻る場合は[return]タグを使います。

*サブルーチン
sub routine
機能的にまとまった処理を行なうルーチン。

●タグ記述例

```
[call storage="ファイル名.ks" target=*ラベル名]¥
[return]¥
[return storage="ファイル名.ks" target=*ラベル名]¥
```

●シナリオ例

```
*chapter2_scene04b|第二話　出発
[eval exp="f.spendtime=0"]¥
[eval exp="f.limittime=120"]¥
[er]¥

(前略)

「さあ行こう！　由美香！」[l]
「どっちへ……？」[p]
```

```
ぼくは、
[delay speed=nowait][locate y=355]¥
[link target=*chapter2_scene06]A．北の道を選んだ[endlink]
[link target=*chapter2_scene09]B．南の道を選んだ[endlink]
[link  target=*chapter2_scene05]C．西側の雑木林に踏み入った
[endlink]¥
[delay speed=user]¥
[s]
;--------------------------------------------
*chapter2_scene05|第二話　西の雑木林
[call target=*chapter2_timechecker]¥
ぼく達は雑木林の中に踏み入った。[l]
外から見ただけではわからなかったが、
立ち枯れた木が至るところに転がっており、
とてもではないが歩けそうになかった。[l]
「無理だ、戻ろう」[l]

　無言の由美香を連れて林を出たぼくは、
もう一度慎重に行く先を検討することにした。[l]

ぼくは、
[delay speed=nowait][locate y=375]¥
[link target=*chapter2_scene06]A．北の道を選んだ[endlink]
[link target=*chapter2_scene09]B．南の道を選んだ[endlink]¥
[delay speed=user]¥
[s]
;--------------------------------------------
*chapter2_scene06|第二話　教会は西
[call target=*chapter2_timechecker]¥
五分ほど歩いたところで道が分かれていた。[l]
西に向かう道、北に向かう道、南に向かう道の三叉路だ。[l]
「教会は西よ……」[l]
背後の由美香がぼんやりと言う。[l]
```

```
ぼくは、
[delay speed=nowait][locate y=355]¥
[link target=*chapter2_scene12]A. 西の道を選んだ[endlink]
[link target=*chapter2_scene07]B. 北の道を選んだ[endlink]
[link target=*chapter2_scene14]C. 南の道を選んだ[endlink]¥
[delay speed=user]¥
[s]
;-------------------------------------------------
*chapter2_scene07|第二話　違う気がする
[call target=*chapter2_timechecker]¥
五分ほど歩いたところで道が分かれていた。[l]
西に向かう道、北に向かう道、南に向かう道の三叉路だ。[l]
「違うような気がする……」
背後の由美香が言う。[l]

ぼくは、
[delay speed=nowait][locate y=355]¥
[link target=*chapter2_scene11]A. 西の道を選んだ[endlink]
[link target=*chapter2_scene08]B. 北の道を選んだ[endlink]
[link target=*chapter2_scene06]C. 南の道を選んだ[endlink]¥
[delay speed=user]¥
[s]
;-------------------------------------------------
*chapter2_timechecker
[eval exp="f.spendtime=f.spendtime+5"]¥
[if  exp="f.spendtime>f.limittime][return  target=*凍死シー
ン][endif]¥
[er]¥
[emb  exp="f.spendtime"]分経過／[emb  exp="f.limittime"]分
[l]
[return]
```

●変数覚書

f.spendtime	経過時間	初期値は0
f.limittime	限界時間	初期値は120

シナリオ例の最初と最後を除く各シーンの冒頭にある[call target=*chapter2_timechecker]で、それぞれの場所から「*chapter2_timechecker」というサブルーチンを呼び出して処理をし、終わったら元の場所に戻っています。

サブルーチン部である「*chapter2_timechecker」については、通常のラベル付きシナリオと同じ扱いで、特に「ここがサブルーチンだよ！」と宣言する必要はありませんが、利用する上でいくつかの注意点があります。

①まず、サブルーチン部は[call]タグでのみアクセスできる場所に書くべきです。KAGのシナリオは上から下に向かって順番にタグを実行するので、気がついたらサブルーチン部も実行されてしまう場合があります。

このような状態を避けるには、サブルーチンの前で[s]と[jump]を使い、サブルーチン部をジャンプして避けるか、サブルーチンは別のファイルにして[call storage="ファイル名.ks" target=*ラベル]で呼び出すようにします。

②もうひとつは、サブルーチンのラベルにセーブ・タイトルを記述してはいけません。サブルーチンは本来見えない処理をする部分のはずですから、そこでセーブされてしまった場合、ロードをした時に予期せぬ状態からゲームが始まることになるからです。

さらに、サブルーチンとして記述したラベルの行末には必ず[return]タグがないといけません。このタグはサブルーチン・ラベルをコールした部分の直後に戻るという大切な働きをしているからです。

シナリオ例ではよく見ると「return」が二つありますが、中ほどの[return]は、経過時間が限界時間を超えた場合に凍死シーンへジャンプするためのもの、下の方の[return]はサブルーチン処理を終えてもとの場所に戻るためのものです。

中ほどの[return]については代わりに[jump]タグを使いたくなりますが、サブルーチン内部でのジャンプはすべて[return]タグで行わないと、コールスタックと呼ばれる値が不正になり、結果としてKAGが正常動作しなくなる場合があります。

1999Christmas-Eveの例では、サブルーチン内に[er]タグを書いておいたため、それぞれのラベルの次行にいちいち[er]¥を置く必要がなくなりました。

このように、まとめられるものは極力まとめていくことが、きれいなシナリオを書く秘訣になります。

6-5 変数のイベント的な利用例

変数や乱数を上手に利用すると、面白いイベントが作れます。

ここでは、3つの扉があり、どれか1つにはお化けがひそんでいるイベントを作ってみます。お化けの隠れている扉は、乱数を利用してシナリオを実行するたびにランダムに変わるようにします。

①使う変数

変 数	内 容	初期値
f.door1	扉1を開けたかどうか	0
f.door2	扉2を開けたかどうか	0
f.door3	扉3を開けたかどうか	0
f.fear	お化けがひそむ扉（1～3でランダム）	

●記述例

```
[eval exp="f.door1=0"]¥
[eval exp="f.door2=0"]¥
[eval exp="f.door3=0"]¥
[eval exp="f.fear=intrandom(1,3)"]¥
```

②変数の変化

扉を一度開けたら変数の値を「1」にして、次からは選べないようにする。

```
[eval exp="f.door1=1"]¥
[eval exp="f.door2=1"]¥
[eval exp="f.door3=1"]¥
```

③変数の値による分岐処理

変数「f.door*」の値が「1」の時はすでに一度開けているため、次からは選択肢を表示しないようにする。ここでは「f.door*」の値が「0」の時のみ選択肢を表示するように指定。

```
[if exp="f.door1==0"][link target=*door1]右の扉[endlink]
[r][endif]¥
[if exp="f.door2==0"][link target=*door2]真中の扉[endlink]
[r][endif]¥
```

```
[if exp="f.door3==0"][link target=*door3]左の扉[endlink]
[r][endif]¥
```

④お化け登場

乱数で得られた番号の扉を開けた時、恐怖が飛び出す。

```
*door1
[er]¥
[if exp="f.fear==1"][jump target=*恐怖][endif]¥

*door2
[er]¥
[if exp="f.fear==2"][jump target=*恐怖][endif]¥

*door3
[er]¥
[if exp="f.fear==3"][jump target=*恐怖][endif]¥
```

●シナリオ記述例

```
*start
;変数の初期設定
[eval exp="f.door1=0"]¥
[eval exp="f.door2=0"]¥
[eval exp="f.door3=0"]¥
[eval exp="f.fear=intrandom(1,3)"]¥
;----------------------------------------
*select
[er]¥
どれを開ける？[l]

[nowait]¥
[if exp="f.door1==0"][link target=*door1]右の扉[endlink]
[r][endif]¥
[if exp="f.door2==0"][link target=*door2]真中の扉[endlink]
[r][endif]¥
```

```
[if exp="f.door3==0"][link target=*door3]左の扉[endlink]
[r][endif]¥
[endnowait]¥
[s]
;-----------------------------------------------
*door1
[er]¥
[if exp="f.fear==1"][jump target=*恐怖][endif]¥
なにもない。[l]
[eval exp="f.door1=1"]¥
[jump target=*select]¥
[s]
;-----------------------------------------------
*door2
[er]¥
[if exp="f.fear==2"][jump target=*恐怖][endif]¥
なにもない。[l]
[eval exp="f.door2=1"]¥
[jump target=*select]¥
[s]
;-----------------------------------------------
*door3
[er]¥
[if exp="f.fear==3"][jump target=*恐怖][endif]¥
なにもない。[l]
[eval exp="f.door3=1"]¥
[jump target=*select]¥
[s]
;-----------------------------------------------
*恐怖
[er]¥
「ぎゃああああああああああああああ！」[l]
(恐怖の絵を見せてやってください) [l]
[s]
```

Step 7
マクロを活用する
KAGのシナリオは、書けば書くほどゴチャゴチャしてきますが、そんな時に便利な機能として準備されているのが「マクロ定義機能」です。ここではマクロの基本的な取り扱いについて解説します。

7-1 マクロの基本的な使い方

KAGの「マクロ」とは、複数のタグを一つのタグにまとめて自由に定義できる機能のことです。たとえば、以下のようなトランジション命令をシナリオ内に記述したとします。

```
[trans time=2000 rule="trans1.png" vague=100]¥
[wt]¥
```

これが一ヶ所だけならいいのですが、シナリオ中に100も200も出てくる場合に、そのたびに2行のタグ命令を記述するのは面倒です。

「コピーすればいいじゃん」と思うかもしれませんが、後からruleファイルを全部別のものに変更したいと思ったり、トランジション・タイムを変更したいと思った時には、記述されたタグを全て修正しなければなりません。

「エディタで一括置換をすれば一発じゃないか」と言う人は、シナリオ・ファイルが100も200もあった場合はどうしますか？

このような時は、上記の2行を[macro][endmacro]タグを用いて、シナリオの冒頭などでマクロ化しておくと便利なのです。

●タグ記述例

```
[macro name=新しいマクロの名前]¥
マクロ化したいタグ行（何行でもよい）
[endmacro]¥
```

●シナリオ例

```
*start
;-------------------------------------------
[macro name=kirikae]¥
[trans time=2000 rule="trans1.png" vague=100]¥
[wt]¥
[endmacro]¥
;-------------------------------------------
*scene01
```

ワンポイント

私の場合、マクロは本シナリオ・ファイル内ではなく「macro.ks」というファイルにすべて記述しておき、それを「first.ks」の冒頭で[call]タグを用いて呼び出すことにしています。

つまり、Step6で説明したサブルーチン化しているんです。

こうすることで、マクロの定義を変更したいと思った場合、長い本シナリオ内を探さなくても「macro.ks」ファイル内を探せばすぐに見付かるので、お勧めです。

この例の場合、[trans]と[wt]の行が[kirikae]という名前の新しいマクロタグに定義されました。以後は[trans][wt]の2行で処理を行なう部分に、[kirikae]というオリジナル・タグを使うことができます。

このようにマクロを使えば、簡単にオリジナルのタグを作れるので非常に便利ですが、使いすぎると困ったことになります。

たとえば、

```
[layopt layer=message0 page=back visible=false]¥
```

のような1行の記述をマクロにする意味はあまりありません。

また、

```
[layopt layer=message0 page=back visible=true]¥
[backlay]¥
```

の場合は、使い方によっては便利ですが、そうでない場合もあります。

なぜなら、特にそれだけで完結していない処理をマクロに定義すると、前後の関係から再度それを分解しなければ処理を変更できないことが多いからです。

完全にシナリオが書き上がった時点で同じ記述がされている行をマクロ化して視認性を高めるならかまいませんが、例のようなケースは「どのような処理を行なっているのか」が分かりにくいため、マクロにしないほうがよいでしょう。

ですが、短いタグでもマクロに定義したほうが便利な場合もあります。

```
[playbgm storage="shock.mid" loop=false]¥
[wb]¥
```

この例では、「shock」という名前の短いMIDIファイルをループさせずに一度だけ鳴らす処理をしていますが、この場合は属性の変更はあり得ないのでマクロ化するメリットがあります。

マクロのメリットで最も大きい点は属性部分の値が完全に統一できることなので、定型処理をマクロに登録しておけば、前回の処理を参考にして属性に同じ値を記述するような面倒を犯す必要はありません。

KAGはオリジナルのタグを一切使わずにすべてをマクロで処理することも許しているので、自分のゲームに見合った処理を行なうためのマクロはどんどん開発して使ってください。

■マクロに属性を与える

下記の例のように、派手にマクロにしたいのに微妙な記述の違いがあって断念したことはありませんか？

●シナリオ例1

```
[layopt layer=message0 page=back visible=false]¥
[image storage="dinner.jpg" layer=base page=back]¥
[trans time=2000 rule="trans01.png" vague=100]¥
[wt]¥
[layopt layer=message0 page=fore visible=true]¥
```

●シナリオ例2

```
[layopt layer=message0 page=back visible=false]¥
[image storage="drive.jpg" layer=base page=back]¥
[trans time=2000 rule="trans01.png" vague=100]¥
[wt]¥
[layopt layer=message0 page=fore visible=true]¥
```

●シナリオ例3

```
[layopt layer=message0 page=back visible=false]¥
[image storage="forest.jpg" layer=base page=back]¥
[trans time=2000 rule="trans01.png" vague=100]
[wt]¥
[layopt layer=message0 page=fore visible=true]¥
```

この例では、①[image]タグで画像をbaseレイヤーのバックグラウンドに読み込み、②2秒でトランジションさせ、③さらにその終了を[wt]で待ってから、④[layopt]タグでメッセージレイヤーのフォアグラウンドを可視状態にする――という一連の処理をしています。ここで異なるのは[image]タグのstorage属性だけですが、このように属性が異なるだけで他の一連の処理が同一の場合も、属性を引き渡すことによってマクロ化することができます。

ただし、異なる指定がされている属性部分はマクロ化できないため、マクロ自体で属性を指定して、それをマクロ内のタグに引き渡さねばなりません。

先の例をマクロ化するには、次のように記述します。

●マクロ定義例

```
[macro name=gamenkirikae]¥
[layopt layer=message0 page=back visible=false]¥
[image storage=%storage layer=base page=back]¥
[trans time=2000 rule="trans01.png" vague=100]¥
[wt]¥
[layopt layer=message0 page=fore visible=true]¥
[endmacro]¥
```

[image]タグのstorage属性の部分に書かれた「%storage」という部分が、新しい[gamenkirikae]タグのstorage属性として利用できます。

このマクロを使って最初の記述例を書き直すと、以下のようになります。

●シナリオ例1

```
[gamenkirikae storage="dinner.jpg"]¥
```

●シナリオ例2

```
[gamenkirikae storage="drive.jpg"]¥
```

●シナリオ例3

```
[gamenkirikae storage="forest.jpg"]¥
```

マクロ定義行で半角の%の次に指定されている文字列がオリジナル・タグの属性として利用できるわけです。

「storage」ではややこしくて嫌だという人は、次のように書いてもかまいません。

●マクロ定義例

```
[macro name=gamenkirikae]¥
[image storage=%gazou layer=base page=back]¥
[trans time=2000 rule="trans01.png" vague=100]¥
[wt]¥
[layopt layer=message0 page=fore visible=true]¥
[endmacro]¥
```

このようにマクロを定義しておくと、次のような記述で新タグが使えます。

●タグ記述例

```
[gamenkirikae gazou="dinner.jpg"]¥
```

この場合は、「gazou」という属性に指定されたファイル名が[image]タグのstorage属性にそのまま渡されます。

また、ルール・ファイルも使うたびに指定したい場合は、次のように定義します。

●マクロ定義例

```
[macro name=gamenkirikae]¥
[image storage=%gazou layer=base page=back]¥
[trans time=2000 ru=%rule vague=100]¥
[wt]¥
[layopt layer=message0 page=fore visible=true]¥
[endmacro]¥
```

●タグ記述例

```
[gamenkirikae gazou="dinner.jpg" ru="trans2.png"]¥
```

このように、マクロ定義タグ[macro][endmacro]を利用すれば、タグだけでなく属性もオリジナルのタグを生成することができ、非常に便利です。

■選択肢をマクロ化する

選択肢は、ノベルやアドベンチャーゲームになくてはならないものです。この部分をマクロ化すると、非常に便利です。

その前に、まず選択肢の記述について考えてみます。次の例は、横書きのノベル形式の選択肢です。

●シナリオ例

```
[er]¥
「言いなさいよ。言わないとサイドブレーキ引くわよ」

[link target=*chapter0_scene14]A．天気がいつもこうだったらいいのに[endlink]
[link target=*chapter0_scene15]B．車の調子がいつもこうだったらいいのに[endlink]
[link target=*chapter0_scene16]C．胃の調子がいつもこうだったらいいのに[endlink]¥
[s]
```

シンプルに書けばこのようになりますが、これでは選択肢としては中途半端です。そこで、選択肢の文字に対して以下のような表現を追加してみます。

①一瞬で表示されるようにする。
②メッセージ履歴に残さないようにする。
③画面の下端に表示する。

①を可能にするには、[nowait][endnowait]タグを用います。

●シナリオ例

```
[er]¥
「言いなさいよ。言わないとサイドブレーキ引くわよ」
[nowait]¥
[link target=*chapter0_scene14]A．天気がいつもこうだったらいいのに[endlink]
[link target=*chapter0_scene15]B．車の調子がいつもこうだったらいいのに[endlink]
[link target=*chapter0_scene16]C．胃の調子がいつもこうだったらいいのに[endlink]¥
[endnowait]¥
[s]
```

さらに②を可能にするため、[history]タグを用います。

●シナリオ例

```
[er]¥
「言いなさいよ。言わないとサイドブレーキ引くわよ」
[nowait]¥
[history output=false]¥
[link target=*chapter0_scene14]A. 天気がいつもこうだったらいいのに[endlink]
[link target=*chapter0_scene15]B. 車の調子がいつもこうだったらいいのに[endlink]
[link target=*chapter0_scene16]C. 胃の調子がいつもこうだったらいいのに[endlink]¥
[history output=true]¥
[endnowait]¥
[s]
```

[history]タグはメッセージ履歴への文字の出力を制御するためのタグで、「output=false」と書いた時点から画面に書かれた文字はメッセージ履歴に出力されなくなります。元に戻すには、再び[history]タグを記述し、「output=true」とします。

③を可能にするため、[locate]タグで位置指定を行ないます。

●シナリオ例

```
[er]¥
「言いなさいよ。言わないとサイドブレーキ引くわよ」
[locate y=365]¥
[nowait]¥
[history output=false]¥
[link target=*chapter0_scene14]A. 天気がいつもこうだったらいいのに[endlink]
[link target=*chapter0_scene15]B. 車の調子がいつもこうだったらいいのに[endlink]
[link target=*chapter0_scene16]C. 胃の調子がいつもこうだったらいいのに[endlink]¥
[history output=true]¥
```

```
[endnowait]¥
[s]
```

これで、3つの選択肢を表示する記述は終了しました。続いてマクロ化してみます。

●マクロ定義例 1

```
[macro name=select3]¥
[locate y=365]¥
[nowait]¥
[history output=false]¥
[endmacro]¥
```

●マクロ定義例 2

```
[macro name=endselect]¥
[history output=true]¥
[endnowait]¥
[endmacro]¥
```

これで[select3][endselect]という2つのオリジナル・タグが利用できるようになりました。このオリジナル・タグを使って元のシナリオを書き換えてみます。

●シナリオ例

```
[er]¥
「言いなさいよ。言わないとサイドブレーキ引くわよ」
[select3]¥
[link target=*chapter0_scene14]A. 天気がいつもこうだったらいいのに[endlink]
[link target=*chapter0_scene15]B. 車の調子がいつもこうだったらいいのに[endlink]
[link target=*chapter0_scene16]C. 胃の調子がいつもこうだったらいいのに[endlink]¥
[endselect]¥
[s]
```

ずいぶんすっきりしたと思います。ただし、このマクロでは、選択肢の数が3つのときにしか使えません。そこで、2つや4つの場合の[select2]や[select4]もそれぞれ定義しておきます。

●マクロ定義例

```
[macro name=select2]¥
[locate y=385]¥
[nowait]¥
[history output=false]¥
[endmacro]¥
```

●マクロ定義例

```
[macro name=select4]¥
[locate y=345]¥
[nowait]¥
[history output=false]¥
[endmacro]¥
```

それぞれ画面下端に配置したいので、[locate]で指定するY座標を文字サイズと行間サイズを考慮に入れて変更します。この値は「Config.tjs」の設定によって変化するので、必ず確認してください。

[endselect]は選択肢がいくつでも共通なので、個々に定義する必要はありません。1つ定義しておけば、すべての場合に利用できます。

●シナリオ例

```
[select2]¥
[link target=*chapter0_scene22]A.「ここで待っているかい？」
[endlink]
[link target=*chapter0_scene23]B.「一緒に行くかい？」
[endlink]¥
[endselect]¥
[s]
```

さらに、選択肢そのものをマクロ化することも有益です。

ボタンやリンクの上にマウス・カーソルが乗ったりクリックしたりすると音が出るソフトがありますが、KAGの[link]タグでも同様のことができます。

●シナリオ例

```
[select2]¥
[link enterse="oto1.wav" clickse="oto2.wav" target=
*chapter0_scene22]A.「ここで待っているかい？」[endlink]
[link enterse="oto1.wav" clickse="oto2.wav" target=
*chapter0_scene23]B.「一緒に行くかい？」[endlink]¥
[endselect]¥
[s]
```

[link]タグの属性に「enterse="カーソルがフォーカスしたときの音ファイル名"」と「clickse="クリックしたときの音ファイル名"」を指定してやれば、この選択肢は音つきになります。

でも、すべての選択肢に上の例のような書き方をしていると、見にくいですよね。そこで、マクロを使います。

●マクロ定義例

```
[macro name=links]¥
[link enterse="oto1.wav" clickse="oto2.wav"
target=%target]¥
[endmacro]¥
```

[link]にsoundを付けたということで、[links]タグを作ってみました。これでシナリオ例を置き換えると、次のようになります。

●シナリオ例

```
[select2]¥
[links target=*chapter0_scene22]A.「ここで待っているかい？」
[endlink]
[links target=*chapter0_scene23]B.「一緒に行くかい？」
[endlink]¥
```

```
[endselect]¥
[s]
```

ここで注意ですが、[links]タグから[link]タグの属性に引き渡す内容は、この場合は「target」のみとなっています。シナリオ内でジャンプするだけならこれで充分ですが、私の場合は、下記のように[links]タグを定義しています。

```
[macro name=links]¥
[link enterse="oto1.wav" clickse="oto2.wav" storage=%storage
 target=%target exp=%exp]¥
[endmacro]¥
```

ジャンプ先のラベルを指定するtarget属性だけでなく、ファイル間ジャンプに利用するstorageと変数を処理するためのexp属性も引き渡すようにしています。

このようにしておけば、いざ「storage」や「exp」を使いたくなったときにも、

```
[links storgae="xmaseve02.ks" target=*chapter2_scene0_set
exp="f.hensuu=1"]¥
```

のように記述できるからです。

このような定義をしておかないと、使えると思ったはずの属性が使えないことになったりするので、注意してください。マクロの属性がうまく効いていないと思ったら、マクロの定義記述が間違っていることを疑うとよいでしょう。

また、マクロの記述データはセーブ・データに保存されるので、マクロを定義する前のセーブ・データをロードした場合、定義したマクロが使えない場合があります。これは、すでに定義したマクロの内容を修正した場合も同様です。

このような場合には、もう一度マクロの定義行を通過する形でリプレイしないと、修正部は発動しません。

ワンポイント

私の場合、「[call storage="macro.ks" target=*start]¥」の1行をエディタのショートカットに登録してあり、必要に応じて呼び出しています。

シナリオを書いていて、もしマクロに修正が入った場合は、①上の1行を本シナリオに仮書き込みしてマクロ定義部分を実行し、②その後セーブデータを保存してから、③上の1行を削除してシナリオを元の状態に戻しています。

シナリオが長くなってくるとfirst.ks部からのリプレイは現実的ではないので、任意の場所からマクロ定義ファイルを呼べる状態にしているわけです。

Step 8 クリッカブル・マップの利用

アドベンチャー形式のゲームのように、画面の一部をクリックすると反応があるような作品を作りたければ、この「クリッカブル・マップ」を利用します。

8-1 クリッカブル・マップの作成方法

「クリッカブル・マップ」とは、画像の一部分をクリックしたらの指定のラベルにジャンプすることを可能にするものです。

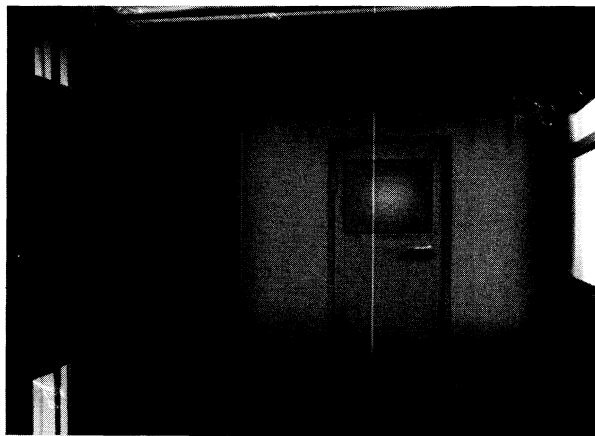
これを「吉里吉里/KAG」で実行するには、以下の3つのファイルが必要です。

ファイル	名前のつけ方
ベース画像	任意のファイル名.拡張子
領域画像	任意のファイル名_p.png
領域アクション定義ファイル	任意のファイル名.ma

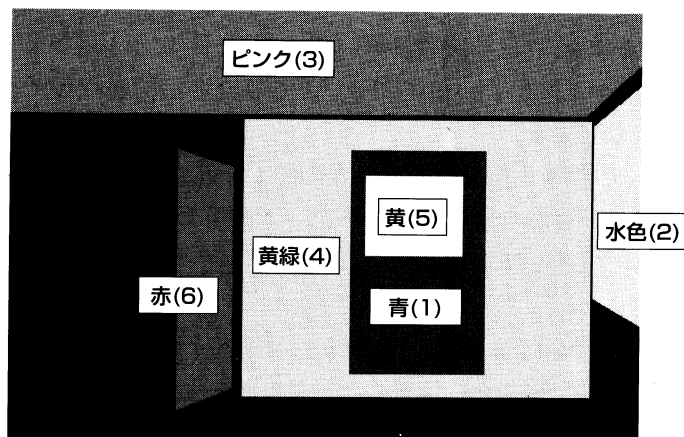
ワンポイント

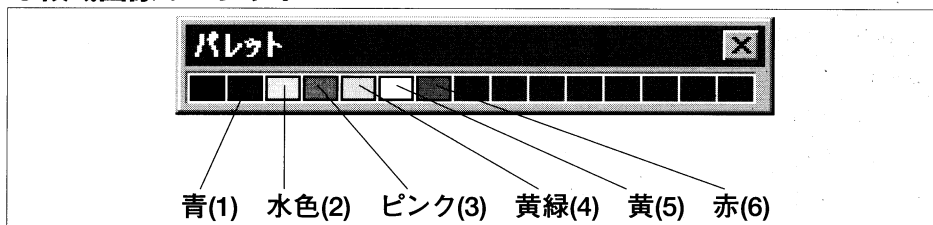
ベース画像の形式は、展開速度を優先するならばBMP、圧縮率を優先するならPNGかJPGがよいでしょう。

≪ベース画像≫

これは、普通の画像ファイルで、「baseレイヤー」「前景レイヤー」のどちらにでも読み込むことができ、ファイル形式も自由です。「前景レイヤー」に読み込んで利用する場合は画像サイズについては任意ですが、「baseレイヤー」に読み込んで利用する場合は「Config.tjs」で指定された画面サイズと等しいものを利用する必要があります。

ベース画像例：exit.png

≪領域画像≫

これは、ベース画像のクリック領域を色分けしたもので、256色のPNGファイルでないといけません。256色が取り扱えるフリーソフトなどを利用してベース画像の形状を元にクリック領域をそれぞれ単色で塗りつぶしていくのですが、このときカラーパレットのインデックス番号が1の色から使う必要があります。

領域画像例：exit_p.png

●領域画像のパレット

この例では、パレットの左側から順番に「0」「1」「2」というパレット・インデックス番号がついています。このため、青は「1」、水色は「2」、ピンクは「3」、黄緑は「4」、黄色は「5」、そして赤が「6」になります。このパレット・インデックス番号は後述の領域アクション定義ファイルに利用するので、きちんと把握しておく必要があります。

≪領域アクション定義ファイル≫

中身は以下のようになっています。

●領域アクション定義ファイル例「exit.ma」

```
1: hint="非常口"; storage="first.ks"; target="*scene01";
2: hint="窓"; storage="first.ks"; target="*scene02";
3: hint="天井"; storage="first.ks"; target="*scene03";
4: hint="奥の壁"; storage="first.ks"; target="*scene04";
5: hint="非常口のガラス"; storage="first.ks";target="*scene05";
6: hint="教室の扉"; storage="first.ks"; target="*scene06";
```

ワンポイント

領域画像を作るには256色専用のペイントソフトが便利です。フリーソフトにはよいものがあるので探してみるといいでしょう。

ワンポイント

パレットの左上が0番の色になります。ここは必ず黒にしておきます。

ワンポイント

左の例ですが、hint=は無理に記述する必要はありません。むしろ記述しない方がゲーム内のクリッカブル・マップらしくてベターです。

それぞれの行の構成は「領域番号（半角の「:」）アクション」で構成されていますが、行頭にある「1」から「6」までの数字が領域画像で塗った色のパレット番号になります。

青色で塗ったのは「非常口の扉」なので**1行目**に、水色で塗ったのは「窓」なので**2行目**に、というように色のパレット番号に合わせてジャンプ先のファイルやラベルを記述していきます。

アクションについてはKAGの説明からはやや外れますが、「属性名＝値；」とう形式でいくつでもつないで書くことができます。

●アクション記述例

storage="ファイル名";	クリックされたときにジャンプするファイル名を指定。
target="ラベル名";	クリックされたときにジャンプするラベル名を指定。
onenter="TJS 式"	領域内にカーソルが入ったときに実行するTJS式を指定。
onleave="TJS 式"	領域内からカーソルが出たときに実行するTJS式を指定。
hint="文字列"	マウスが乗ったときに表示される文字列を記述。
exp="TJS式"	変数処理や、TJS機能の呼び出しなどのTJS式を指定。
cursor="ファイル/定数名"	マウスカーソル・ファイルやマウスカーソル定数を指定。

以上の3つのファイルを同一のフォルダに入れれば、クリッカブル・マップの準備は完了です。背景画像をクリッカブル・マップとして利用するなら「bgimage」フォルダへ、前景画像なら「fgimage」フォルダへ入れましょう。「other」でもかまいません。

続いてシナリオ・ファイルでクリッカブル・マップを呼び出す記述を行ないますが、これは [image]タグで行ないます。ベース画像と同名の「_p.png」と.maファイルが存在すれば、ベースとなる画像を[image]タグで普通に呼び出しただけでクリッカブル・マップが利用できます。

●タグ記述例

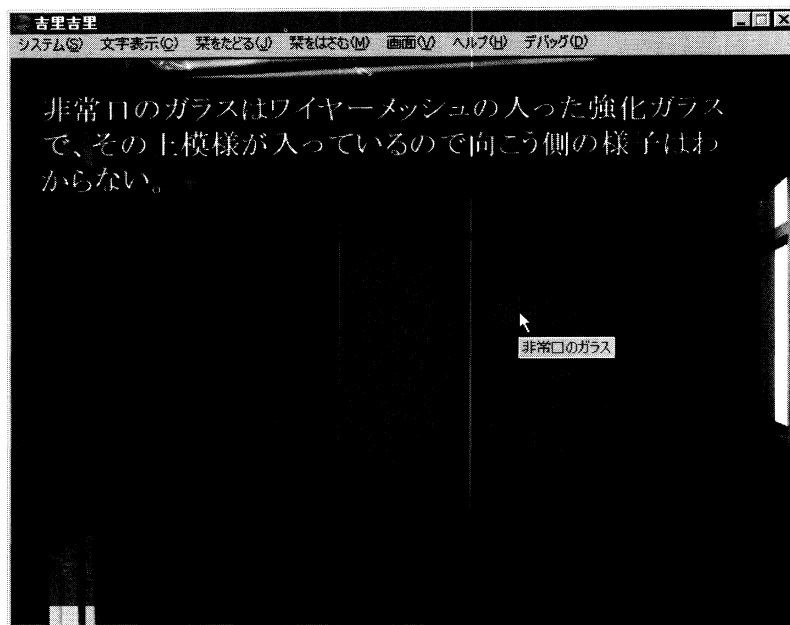
```
[image layer=base page=fore storage="exit.png"]¥
```

●シナリオ記述例

```
*start
[er]¥
[image layer=base page=fore storage="exit.png"]¥
[s]
;---------------------
*scene01
[er]¥
非常口は鍵が掛かっていて、押しても引いても開かなかった。[l]
[jump target=*start]¥
[s]
;---------------------
*scene02
[er]¥
窓からは外の光がうっすらと差し込んでいる。[l]
[jump target=*start]¥
[s]
;---------------------
*scene03
[er]¥
天井……そこは調べようがない。[l]
[jump target=*start]¥
[s]
;---------------------
*scene04
[er]¥
非常口の扉がついている奥の壁はコンクリートでできていた。[l]
[jump target=*start]¥
[s]
;---------------------
*scene05
[er]¥
非常口のガラスはワイヤーメッシュの入った強化ガラスで、その上模様が入
っているので向こう側の様子はわからない。[l]
```

```
[jump target=*start]¥
[s]
;---------------------
*scene06
[er]¥
先ほど来た時にひどい目に遭った3年2組の教室へ入る扉だ。[l]
[jump target=*start]¥
[s]
```

コラム～少◯のひとりごと

クリッカブル・マップはアドベンチャーゲームでは必須の機能ですが、シナリオに[link]タグが1つもないのに分岐処理が行なえるので、ちょっと不思議な感じがしませんか？

実行例

■クリッカブル・マップを有効のまま利用する

先ほどの例では、それぞれのジャンプ先から[jump]タグを使ってクリッカブル・マップ表示部分に戻っていましたが、これは「領域番号0」を使った特殊な指定をすることにより、シナリオを大幅に改善することができます。

具体的な方法ですが、領域アクション定義ファイル内に「領域番号0」を記述し、次のように記述します。

●領域アクション定義ファイル例「exit.ma」

```
0: autodisable=false;
1: hint="非常口"; storage="first.ks"; target="*scene01";
2: hint="窓"; storage="first.ks"; target="*scene02";
3: hint="天井"; storage="first.ks"; target="*scene03";
4: hint="奥の壁"; storage="first.ks"; target="*scene04";
5: hint="非常口のガラス"; storage="first.ks"; target=
   "*scene05";
6: hint="教室の扉"; storage="first.ks"; target="*scene06";
```

1行目に「領域番号0」の行を加えることにより、クリッカブル・マップをクリックしても、クリッカブル・マップは無効になりません。

先の例ではこの指定を行なわなかったため、それぞれのジャンプ先から再びラベル「*start」へ戻り、[image]タグによるクリッカブル・マップ画像のロードを行なっていましたが、「領域番号0」の行を加えることで不要になります。

●シナリオ記述例

```
*start
[er]¥
[image layer=base page=fore storage="exit.png"]¥
[s]
;---------------------
*scene01
[er]¥
非常口は鍵が掛かっていて、押しても引いても開かなかった。[l]
[s]
;---------------------
*scene02
[er]¥
窓からは外の光がうっすらと差し込んでいる。[l]
[s]
;---------------------
*scene03
[er]¥
```

```
天井……そこは調べようがない。[l]
[s]
;---------------------
*scene04
[er]¥
非常口の扉がついている奥の壁はコンクリートでできていた。[l]
[s]
;---------------------
*scene05
[er]¥
非常口のガラスはワイヤーメッシュの入った強化ガラスで、その上模様が入
っているので向こう側の様子はわからない。[l]
[s]
;---------------------
*scene06
[er]¥
先ほど来た時にひどい目に遭った３年２組の教室へ入る扉だ。[l]
[s]
```

このように[jump]タグをそれぞれのジャンプ先に置かなくても、常に画像が表示されている間はクリッカブル・マップが有効になっているため、別の領域をクリックすることができるのです。

■クリッカブル・マップの条件分岐

[if]タグと同様に、クリッカブル・マップでも変数などの条件によって有効にしたり無効にしたりできます。

基本的にクリッカブル・マップではアクションが何も定義されていなければその領域は無視されます。これを応用して、一度クリックして調べた場所は次からクリックできないようにしてみます。

●exit.ma記述例

```
0 : autodisable=false;
1:  if(f.sc01!=1){hint="非常口"; storage="first.ks";target
    ="*scene01";}
2:  if(f.sc02!=1){hint="窓"; storage="first.ks";target
    ="*scene02";}
3:  if(f.sc03!=1){hint="天井"; storage="first.ks";target
    ="*scene03";}
4:  if(f.sc04!=1){hint="奥の壁"; storage="first.ks";target
    ="*scene04";}
5:  if(f.sc05!=1){hint="非常口のガラス"; storage="first.ks";
    target="*scene05";}
6:  if(f.sc06!=1){hint="教室の扉"; storage="first.ks";
    target="*scene06";}
```

●シナリオ例

```
;---------------------
*start
[er]¥
[image layer=base page=fore storage="exit.png"]¥
[s]
;---------------------
*scene01
[er]¥
[eval exp="f.sc01=1"]¥
非常口は鍵が掛かっていて、押しても引いても開かなかった。[l]
[s]
;---------------------
*scene02
[er]¥
[eval exp="f.sc02=1"]¥
窓からは外の光がうっすらと差し込んでいる。[l]
[s]
;---------------------
```

```
*scene03
[er]¥
[eval exp="f.sc03=1"]¥
天井……そこは調べようがない。[l]
[s]
;---------------------
*scene04
[er]¥
[eval exp="f.sc04=1"]¥
非常口の扉がついている奥の壁はコンクリートでできていた。[l]
[s]
;---------------------
*scene05
[er]¥
[eval exp="f.sc05=1"]¥
非常口のガラスはワイヤーメッシュの入った強化ガラスで、その上模様が入
っているので向こう側の様子はわからない。[l]
[s]
;---------------------
*scene06
[er]¥
[eval exp="f.sc06=1"]¥
先ほど来た時にひどい目に遭った3年2組の教室へ入る扉だ。[l]
[s]
```

領域アクション定義の行で、if(f.変数名!=1){アクション}という記述をすることにより、「f.変数名」が「1」でない場合は{アクション}を実行するようにしてあります。

そして、それぞれのジャンプ先で[eval]タグにより「f.変数名」に「1」が代入されることによって、再びKAGシナリオが領域アクション定義の行を参照しても{アクション}は実行されなくなるわけです。

■クリッカブル・マップを使い終えたら

クリッカブル・マップを伴う画像が存在していると、シナリオはキーボードによるマウスエミュレーション・モードで実行されます。

これには弊害があり、edit タグのようなキーボードを用いるほかの機能が正常に動作しなくなるので、クリッカブル・マップは使用を終えた後できちんと片付ける必要があります。

●シナリオ例

```
[mapdisable layer=base page=fore]¥
```

base レイヤーに読み込んだ背景画像は上のような記述でクリッカブル・マップを無効化しておきます。

前景レイヤーに読み込んだ場合は、その画像を引き続き使う必要がないなら、[freeimage]タグを利用してクリッカブル・マップを画像もろとも破棄します。

●シナリオ例

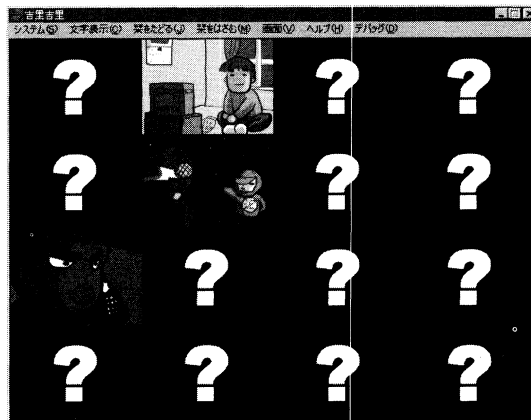
```
[freeimage layer=0 page=fore]¥
[freeimage layer=0 page=back]¥
```

この[freeimage]タグはアニメーションする前景画像を開放する場合にも利用できます。特にアニメーション・ファイルは「page=back」などに置いたままにしておくとメモリを無駄に消費するので、必ずこのタグを用いて開放しておきましょう。

また、クリッカブル・マップを持つ画像がロードされているレイヤーに別の画像をロードしてもクリッカブル・マップは解放されて消えます。

クリッカブル・マップをbase レイヤーに背景として読み込んでいる場合は、次に表示する背景画像を読み込めばクリッカブル・マップは自動的に無効になるので、トラブルは少ないです。しかし、前景レイヤーに使っている場合には解放を忘れることがあるので、注意してください。

8-2 クリッカブル・マップを応用したCGギャラリー

クリッカブル・マップの機能を応用することにより、一度見た画像はサムネイル表示し、見ていない画像は「？」の描かれた真っ黒な矩形などで表示するCGギャラリーを作ることができます。もちろん、サムネイルをクリックすると原寸のCGを表示するものです。

このようなCGギャラリーを作成するには、以下のような処理をシナリオ化することが必要になります。

①本シナリオ内で任意のCGを見た時、それを変数に記憶。
②CGギャラリーのメニューで変数をチェックし、見たCGのみをサムネイル表示。

ここまでは、変数と[if]分岐を利用すれば組み上げることができます。

もし、扱われているイベントCGの枚数があまりにも多くて変数を1つずつ定義するのが面倒な場合は、変数名を直接CGファイル名から得ることもできます。

この場合、一度でもゲーム内で表示した（つまりプレイヤーが見た）画像を記録するために以下のようなマクロを定義し、CGアルバムに掲載する画像は通常の[image]タグではなく、このオリジナル・タグで表示します。

●マクロ定義例

```
[macro name=imagemem]¥
[image *]¥
[eval exp="sf[mp.storage]=1"]¥
[endmacro]¥
```

この[image]タグに毛が生えたような[imagemem]タグを利用してゲーム内でCGを表示すると、下のシナリオ例の２行目にコメントとして記述されているような[eval]タグを用いた変数定義が自動的に行なわれます。

変数名には[imagemem]タグでロードした画像データのファイル名が利用され、その値には自動的に「1」が入ります。

●シナリオ例

```
[imagemem storage="pic01.jpg" layer=base page=fore]¥
;[eval exp="sf.pic01=1"]¥
```

ここまでで、シナリオ内で表示した画像については、画像を表示するとその画像ファイル名をもつ変数が生成され、値に「1」が代入されることになりました。

続いて、CGギャラリーのメニューページを作りますが、考え方は以下のようになります。

①すべてのイベントCGのサムネイルを並べた１枚の画像を作成し、baseレイヤーのフォアグラウンドにロードする。
②[laycount]タグにより、イベントCGの数だけ前景レイヤーを準備する。
③それぞれの前景レイヤーに「まだ見ていません」というサムネイルと同サイズの画像を読み込み、baseレイヤー上の個々のサムネイルを覆い隠す。
④それぞれの前景レイヤー上に読み込まれた「まだ見ていません」という画像を、シナリオ内部で得た変数の値を元に管理する。具体的には、変数が「0」の場合は覆い隠したままにしておき、「1」の場合には取り払う。
⑤「まだ見ていません」画像が取り払われた場合、クリッカブル・マップが有効になり、baseレイヤー上のサムネイルをクリックすると、そのサムネイルからリンクしている原寸の元画像を画面に表示する。

[laycount]は、メッセージ・レイヤーや前景レイヤーなど、動的に数が変更される可能性のあるレイヤーの現在の利用数を指定するためのタグです。

ワンポイント

CGギャラリーを作るために必要な前景レイヤーの数はbaseレイヤーに並べて１枚絵として表示したサムネイルの総数に等しくなります。

たとえば４×4=16枚のCGサムネイル画面を作る場合には前景レイヤーは16枚必要になりますが、この程度ならば特に問題はありません。

W.Deeさんは「サイズが小さい前景レイヤーなら、数十枚同時に利用してもさほど問題にならない」と言っています。

ワンポイント

[laycount]タグでメッセージレイヤー数を指定する場合、message=0という指定はできません。メッセージレイヤーは最低でも常に1枚存在する状態でなければならないからです。

●記述例

```
[laycount messages=2 layers=10]¥
[laycount messages=1 layers=0]¥
```

上の例ではメッセージ・レイヤーを2枚と前景レイヤーを10枚準備しており、下の例ではメッセージ・レイヤーを最小数の1枚にし、前景レイヤーはすべて破棄しています。

また、メッセージ・レイヤーのみ、前景レイヤーのみを指定することもできます。

●記述例

```
[laycount messages=2]¥
[laycount layers=10]¥
```

以下に、16×16のサムネイルを「？」の画像で隠しておき、シナリオ内で見た画像だけサムネイルを表示するシナリオを書いてみます。

●シナリオ例

```
*start
[position layer=message0 left=0 top=0 width=640
height=480 opacity=150]¥
[layopt layer=message0 page=fore visible=true]¥
;
;すべての変数を0にします。
[eval exp="sf.pic01=0"]¥
[eval exp="sf.pic02=0"]¥
[eval exp="sf.pic03=0"]¥
[eval exp="sf.pic04=0"]¥
[eval exp="sf.pic05=0"]¥
[eval exp="sf.pic06=0"]¥
[eval exp="sf.pic07=0"]¥
[eval exp="sf.pic08=0"]¥
[eval exp="sf.pic09=0"]¥
[eval exp="sf.pic10=0"]¥
[eval exp="sf.pic11=0"]¥
```

```
[eval exp="sf.pic12=0"]¥
[eval exp="sf.pic13=0"]¥
[eval exp="sf.pic14=0"]¥
[eval exp="sf.pic15=0"]¥
[eval exp="sf.pic16=0"]¥
;
;以下の行は本来なら各シナリオでCGを見た時に1になるシステム変数です。
;ここではデバッグの意味もあり、16個の変数を手動で操作します。
;本来はこれらの変数は先に定義した[imagemem]タグで管理します。
;見たことにしたい画像の行から先頭のセミコロンを外してみてください。
;[eval exp="sf.pic01=1"]¥
[eval exp="sf.pic02=1"]¥
;[eval exp="sf.pic03=1"]¥
;[eval exp="sf.pic04=1"]¥
;[eval exp="sf.pic05=1"]¥
[eval exp="sf.pic06=1"]¥
[eval exp="sf.pic07=1"]¥
[eval exp="sf.pic08=1"]¥
;[eval exp="sf.pic09=1"]¥
;[eval exp="sf.pic10=1"]¥
[eval exp="sf.pic11=1"]¥
;[eval exp="sf.pic12=1"]¥
;[eval exp="sf.pic13=1"]¥
;[eval exp="sf.pic14=1"]¥
[eval exp="sf.pic15=1"]¥
;[eval exp="sf.pic16=1"]¥
[link target=*cg_room]CGギャラリーへ行く[endlink]¥
[s]
;---------------------------------------------
*cg_room
[er]¥
;
;16×16のサムネイルが並べられた1枚の画像をbaseレイヤーに表示します。
[image storage="allpic.bmp" layer=base page=fore]¥
```

```
;
;前景レイヤーを17枚にします。
[laycount layers=17]¥
;
;全ての前景レイヤーを見えるようにします。
[layopt layer=0 page=fore visible=true]¥
[layopt layer=1 page=fore visible=true]¥
[layopt layer=2 page=fore visible=true]¥
[layopt layer=3 page=fore visible=true]¥
[layopt layer=4 page=fore visible=true]¥
[layopt layer=5 page=fore visible=true]¥
[layopt layer=6 page=fore visible=true]¥
[layopt layer=7 page=fore visible=true]¥
[layopt layer=8 page=fore visible=true]¥
[layopt layer=9 page=fore visible=true]¥
[layopt layer=10 page=fore visible=true]¥
[layopt layer=11 page=fore visible=true]¥
[layopt layer=12 page=fore visible=true]¥
[layopt layer=13 page=fore visible=true]¥
[layopt layer=14 page=fore visible=true]¥
[layopt layer=15 page=fore visible=true]¥
;
;それぞれの前景レイヤーに「?」マークの画像を置きます。
;この画像はシステム変数によって表示されたりされなかったりします。
[image storage="notyet.bmp" layer=0 page=fore left=0
top=0 cond="sf.pic01==0"]¥
[image storage="notyet.bmp" layer=1 page=fore left=160
top=0 cond="sf.pic02==0"]¥
[image storage="notyet.bmp" layer=2 page=fore left=320
top=0 cond="sf.pic03==0"]¥
[image storage="notyet.bmp" layer=3 page=fore left=480
top=0 cond="sf.pic04==0"]¥
[image storage="notyet.bmp" layer=4 page=fore left=0
top=120 cond="sf.pic05==0"]¥
```

```
[image storage="notyet.bmp" layer=5 page=fore left=160
top=120 cond="sf.pic06==0"]¥
[image storage="notyet.bmp" layer=6 page=fore left=320
top=120 cond="sf.pic07==0"]¥
[image storage="notyet.bmp" layer=7 page=fore left=480
top=120 cond="sf.pic08==0"]¥
[image storage="notyet.bmp" layer=8 page=fore left=0
top=240 cond="sf.pic09==0"]¥
[image storage="notyet.bmp" layer=9 page=fore left=160
top=240 cond="sf.pic10==0"]¥
[image storage="notyet.bmp" layer=10 page=fore left=320
top=240 cond="sf.pic11==0"]¥
[image storage="notyet.bmp" layer=11 page=fore left=480
top=240 cond="sf.pic12==0"]¥
[image storage="notyet.bmp" layer=12 page=fore left=0
top=360 cond="sf.pic13==0"]¥
[image storage="notyet.bmp" layer=13 page=fore left=160
top=360 cond="sf.pic14==0"]¥
[image storage="notyet.bmp" layer=14 page=fore left=320
top=360 cond="sf.pic15==0"]¥
[image storage="notyet.bmp" layer=15 page=fore left=480
top=360 cond="sf.pic16==0"]¥
[link target=*return_start]メニューに戻る[endlink]¥
[s]
;
;メニューに戻るために前景レイヤーを0にし、黒い画像をbaseに読み込み
ます。
*return_start
[er]¥
[laycount layers=0]¥
[image storage="black.png" layer=base page=fore]¥
[jump target=*start]¥
[s]
;
```

ワンポイント

[image]タグの末尾にあるcond属性は、[if]タグと同じ働きをします。「cond=」の後ろに書かれた評価式が"真"の場合、つまり式を満たしているときのみ、cond属性が記述されているタグが実行されるようになります。

この場合は"sf.pic0x==0"ですから、指定のイベントCGを見ていない場合に[image]タグが実行され、「？」マークの描かれた黒い矩形が表示されてサムネイルを覆い隠しています。

cond属性はほとんどのタグに利用できるので、[if]タグを充分に理解できたら使ってみるとよいでしょう。

```
;それぞれのクリッカブルマップがクリックされた時にジャンプする先です。
;本来はここで原寸のCGを表示します。
*pshow01
[er]¥
1番目の画像がクリックされました。
[s]
*pshow02
[er]¥
2番目の画像がクリックされました。
[s]
*pshow03
[er]¥
3番目の画像がクリックされました。
[s]
*pshow04
[er]¥
4番目の画像がクリックされました。
[s]
*pshow05
[er]¥
5番目の画像がクリックされました。
[s]
*pshow06
[er]¥
6番目の画像がクリックされました。
[s]
*pshow07
[er]¥
7番目の画像がクリックされました。
[s]
*pshow08
[er]¥
8番目の画像がクリックされました。
[s]
```

```
*pshow09
[er]¥
9番目の画像がクリックされました。
[s]
*pshow10
[er]¥
10番目の画像がクリックされました。
[s]
*pshow11
[er]¥
11番目の画像がクリックされました。
[s]
*pshow12
[er]¥
12番目の画像がクリックされました。
[s]
*pshow13
[er]¥
13番目の画像がクリックされました。
[s]
*pshow14
[er]¥
14番目の画像がクリックされました。
[s]
*pshow15
[er]¥
15番目の画像がクリックされました。
[s]
*pshow16
[er]¥
16番目の画像がクリックされました。
[s]
```

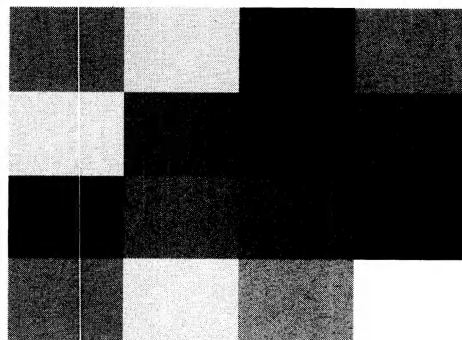
続いて領域アクション定義ファイルの内容です。

●領域アクション定義ファイル「allpic.ma」記述例

```
0: autodisable=false;
1: if(sf.pic01==1){storage="first.ks"; target="*pshow01";}
2: if(sf.pic02==1){storage="first.ks"; target="*pshow02";}
3: if(sf.pic03==1){storage="first.ks"; target="*pshow03";}
4: if(sf.pic04==1){storage="first.ks"; target="*pshow04";}
5: if(sf.pic05==1){storage="first.ks"; target="*pshow05";}
6: if(sf.pic06==1){storage="first.ks"; target="*pshow06";}
7: if(sf.pic07==1){storage="first.ks"; target="*pshow07";}
8: if(sf.pic08==1){storage="first.ks"; target="*pshow08";}
9: if(sf.pic09==1){storage="first.ks"; target="*pshow09";}
10: if(sf.pic10==1){storage="first.ks"; target="*pshow10";}
11: if(sf.pic11==1){storage="first.ks"; target="*pshow11";}
12: if(sf.pic12==1){storage="first.ks"; target="*pshow12";}
13: if(sf.pic13==1){storage="first.ks"; target="*pshow13";}
14: if(sf.pic14==1){storage="first.ks"; target="*pshow14";}
15: if(sf.pic15==1){storage="first.ks"; target="*pshow15";}
16: if(sf.pic16==1){storage="first.ks"; target="*pshow16";}
```

システム変数が「1」のときにfirst.ks内のそれぞれのラベルにジャンプする処理を指定しています。

あとは、サムネイルの並んだベース・ファイルとクリッカブル・マップの領域定義ファイルを用意すれば完成です。

ベース・ファイル

領域定義ファイル

Step 9

その他の機能

スタッフロールの作り方や、雪や雨を降らせるなど、KAG3で提供されている面白いプラグインの利用方法を中心に、さまざまな機能を解説します。

9-1 スキップ状態を制限する

「次の選択肢/未読まで進む」コマンドは、キーボードのFキーを押すことでも実行できます。

「吉里吉里」のシステム・メニューには、「次の選択肢/未読まで進む」というコマンドがあります。これをクリックすると、一度見た文章や画像はウエイトのない状態で高速に表示されます。また、メッセージが表示されている途中で画面をクリックすることによって、クリック待ち部分まで一気に文章を表示する機能もあります。これらの状態を「スキップ状態」と言います。

KAGのほとんどのタグは、このスキップ状態を許可しています。これは「ユーザーを待たせるような処理については、入力があった場合はどんどんスキップする」という仕様が標準になっているからです。

ですが、作品を作っていると、ヒントなどが隠された重要な画像を表示するためのトランジションや、制限時間がついている選択肢など、ユーザーが処理をスキップしてしまうと都合が悪いものもあります。

このような状況下でスキップ状態を強制的に止めたい場合は、[cancelskip]というタグを用います。

≪キャンセル・スキップ[cancelskip]≫

以下は、トランジションの手前で強制的にスキップ状態を解除するシナリオ例です。

●シナリオ例

```
;●背景画像切り替え用のマクロ
[macro name=kirikae]¥
[cancelskip]¥
[image storage=%storage layer=base page=back]¥
[trans method=crossfade time=1000]¥
[wt]¥
[layopt layer=message0 page=fore visible=true]¥
[endmacro]¥
```

このようにマクロ化しておけば、数あるトランジションの手前に[cancelskip]タグを1つずつ書く手間が省けます。

≪クリック・スキップ[clickskip]≫

[cancelskip]によって一度は解除されたスキップ状態も、ユーザーが再度「次の選択肢/未読まで進む」コマンドを選んだり、エンター・キー押下などのアクションを起こした場合には、再びスキップ状態に入ってしまいます。このようなことを避けるためには、スキップ機能をいったん停止させるだけではなく、その機能自体を無効にしてしまえばよいのです。このようなときに利用するのが、[clickskip]タグです。

●タグ記述例

```
[clickskip enabled=false]¥
[clickskip enabled=true]¥
```

上の例でクリック・スキップ自体を禁止し、下の例で再びクリック・スキップできるように許可しています。クリック・スキップの禁止状態は同じ[clickskip]タグでないと解除できないので、「enabled=false」で禁止にした後は、必ず「enabled=true」で解除します。

●シナリオ例

```
;●背景画像切り替え用のマクロ
[macro name=kirikae]¥
[cancelskip]¥
[clickskip enabled=false]¥
[image storage=%storage layer=base page=back]¥
[trans method=crossfade time=1000]¥
[wt]¥
[layopt layer=message0 page=fore visible=true]¥
[clickskip enabled=true]¥
[endmacro]¥
```

このようにしておけば、どのようなモードでトランジションまで読み進めてきても自動的にいったん停止し、トランジションが終了するまではクリック・スキップが無効になります。

9-2 スタッフロールを作る

ここでは、映画やドラマの最後に流れる「キャストロール」や「スタッフロール」を作ってみます。「吉里吉里/KAG」には、これらスタッフロールを利用するための専用プラグイン・サンプルが提供されているので、これを利用すればすぐにできます。

≪プラグイン≫

「KAGスタッフロール・プラグイン」は、「KAG3¥KAG3plugin¥staffroll¥scenario¥」の中にある「staffroll.ks」が本体です。まずはこれを自分のプロジェクト・フォルダ下部のscenarioフォルダにコピーしておきましょう。

≪シナリオの書き方≫

続いて、シナリオの書き方ですが、スタッフロール・プラグインを使いたいプロジェクトの「first.ks」の先頭行に以下のように記述します。

●シナリオ記述例

```
@call storage="staffroll.ks"
```

勘のいい方は気づいたと思いますが、この機能はサブルーチン形式で提供されています。「staffroll.ks」の中を開いてみると、いちばん最後の行に「@return」と書かれていることからも分かります。このサブルーチンを呼び出すことで、以下の5つのスタッフロール用タグが使えるようになります。

staffrollinit	スタッフロールを初期化する
staffrolltext	文字を表示する
staffrollimage	画像を表示する
staffrollstart	スタッフロールのスクロールを開始する
staffrolluninit	スタッフロールを終了する

コラム～少年のひとりごと

「吉里吉里1/KAG2」の時代には[move]タグを利用した非常に複雑なシナリオを書かねば、スタッフロールは作れなかったのです。
あの時代のことを思うと、今はずいぶん楽になりました。

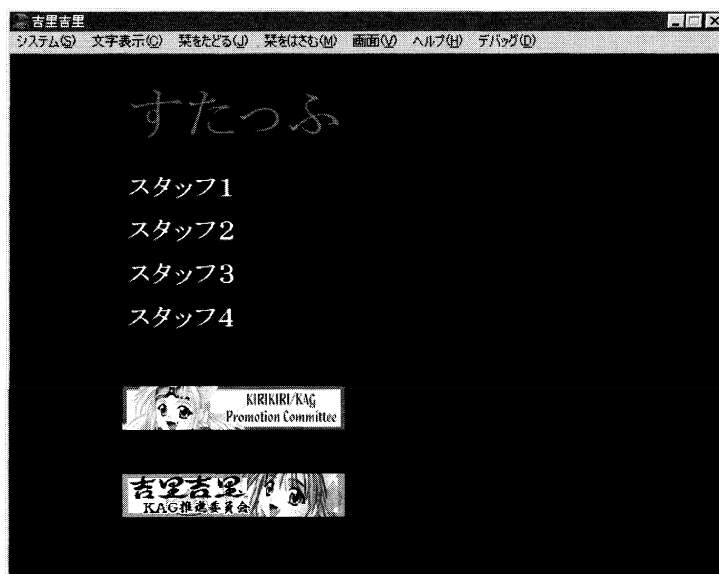
ワンポイント

「staffroll.ks」は2回以上呼び出さないでください。マクロの定義と同じで、1回呼び出せば、それ以降機能が使用可能になります。

ワンポイント

スタッフロールにメッセージ・レイヤーは使いません。このためスタッフロールを使う前にメッセージ・レイヤーを非表示にしておくとよいでしょう。

●シナリオ例

```
@staffrollinit
@font size=50 color=0xff0000
@staffrolltext x=100 y=480 text="すたっふ"
@resetfont
@staffrolltext x=100 y=80 text="スタッフ１"
@staffrolltext x=100 y=40 text="スタッフ２"
@staffrolltext x=100 y=40 text="スタッフ３"
@staffrolltext x=100 y=40 text="スタッフ４"
@staffrollimage x=100 y=80 storage="ayari_32.png"
@staffrollimage x=100 y=80 storage="kiri_s32.png"
@staffrollstart height=1500 time=20000
@wait time=20000
@staffrolluninit
```

実行例

「@staffrolltext」行と「@staffrollimage」行の「x」と「y」や、「@staffrollstart」行の「height」「time」、「@wait」行の「time=20000」の値をいろいろと調整して、自分の作品に合ったスタッフロールを作ってみてください。

ワンポイント

吉里吉里プラグインとKAGプラグインは別のものです。吉里吉里プラグインはDLL形式で提供され、吉里吉里そのものの機能を拡張しますが、KAGプラグインはシナリオの書き方によって実行できる特別な機能を提供するシナリオファイルです。

吉里吉里プラグインは、吉里吉里フォルダ直下のpluginフォルダに、KAGプラグインはKAG3フォルダ直下のkag3pluginフォルダに入っています。

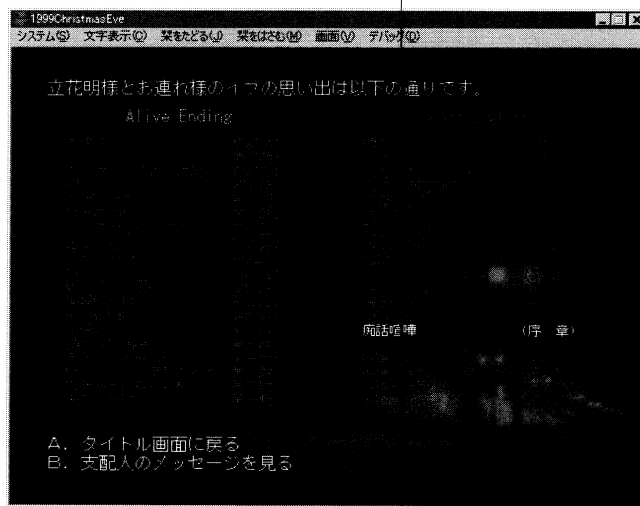
9-3 エンディング・リストを作る

見たエンディング名を白で表示し、まだ見ていないエンディングを灰色で表示するようなエンディング・リストを作るには、クリッカブル・マップのところで使ったシステム変数を用います。

まず、シナリオ内のエンディング記述部分に、[eval]タグでシステム変数を記述します。値は「1」を代入しておきます。

●シナリオ例

```
「一晩中、守ってくれて、ありがとう。明」[p]
[wait time=2000]¥
[scenechange storage="00"]¥
[eval exp="sf.aend17=1"]¥
[jump storage="xmaseve_credit.ks" target=*ending02]¥
[s]
```

シナリオ例4行目の[eval]で指定されている「sf.aend17」がシステム変数であり、aend17というシステム変数を「1」にしています。同様の記述をしておけば、エンディングを見たときにそれぞれの変数が「1」になり、見ていないエンディングは「0」のままなので、あとは、エンディング・リストでそれぞれのシステム変数をチェックして[if]による条件分岐を書けば、完成です。

```
*memories
;[position layer=message0 color=0xFF0000 opacity=0 left=0
top=0 width=640 height=480 marginL=35 marginT=30
marginR=35 marginB=25]¥
[position layer=message1 color=0xFF0000 opacity=0 left=0
top=0 width=640 height=480 marginL=35 marginT=30
marginR=35 marginB=25]¥
[playbgm storage="piano"]¥
[scenechange storage="title2"]¥
[layopt layer=message1 page=fore visible=false]¥
[layopt layer=message1 page=back visible=false]¥
[er]¥
[current layer=message0]¥
[backtxtclear]¥
[layopt layer=message0 page=fore visible=false]¥
[delay speed=nowait]¥
[font color=0xFFFF00]¥
お客様とお連れ様のイブの思い出は以下の通りです。
[locate x=80 y=30][font color=0x00FFFF]Alive Ending
[locate x=380 y=30][font color=0xFF0000]Death Ending
[font size=14 color=0x666666]¥
[locate x=20 y=60][ch text="永遠に……　　　　　　（最終話）"]
[if exp="sf.aend01==1"][locate x=20 y=60][font
color=0xFFFFFF][ch text="永遠に……　　　　　　（最終話）
"][font color=0x666666][r][endif]¥
[locate x=20 y=75][ch text="ホワイトイブ　　　　　（最終話）"]
[if exp="sf.aend02==1"][locate x=20 y=75][font
color=0xFFFFFF][ch text="ホワイトイブ　　　　　（最終話）
"][font color=0x666666][r][endif]¥
[locate x=20 y=90][ch text="ここはどこ？ぼくは誰？（最終話）"]
[if exp="sf.aend03==1"][locate x=20 y=90][font
color=0xFFFFFF][ch text="ここはどこ？ぼくは誰？（最終話）
"][font color=0x666666][r][endif]¥
[locate x=20 y=105][ch text="帰りたい故郷へ　　　　（第七話）"]
[if exp="sf.aend04==1"][locate x=20 y=105][font
color=0xFFFFFF][ch text="帰りたい故郷へ　　　　（第七話）
"][font color=0x666666][r][endif]¥
[locate x=20 y=120][ch text="ロザリオの少女　　　　（第六話）"]
```

```
[if  exp="sf.aend05==1"][locate  x=20  y=120][font
color=0xFFFFFF][ch  text="ロザリオの少女        (第六話)
"][font color=0x666666][r][endif]¥
[locate x=20 y=135][ch text="彼女の神性        (第六話)"]
[if  exp="sf.aend06==1"][locate  x=20  y=135][font
color=0xFFFFFF][ch  text="彼女の神性        (第六話)
"][font color=0x666666][r][endif]¥
[locate x=20 y=150][ch text="一人の力は弱くても   (第六話)"]
[if  exp="sf.aend07==1"][locate  x=20  y=150][font
color=0xFFFFFF][ch  text="一人の力は弱くても   (第六話)
"][font color=0x666666][r][endif]¥
[locate x=20 y=165][ch text="虚ろな日々        (第六話)"]
[if  exp="sf.aend08==1"][locate  x=20  y=165][font
color=0xFFFFFF][ch  text="虚ろな日々        (第六話)
"][font color=0x666666][r][endif]¥
[locate x=20 y=180][ch text="そんなものさ       (第五話)"]
[if  exp="sf.aend09==1"][locate  x=20  y=180][font
color=0xFFFFFF][ch  text="そんなものさ       (第五話)
"][font color=0x666666][r][endif]¥
[locate x=20 y=195][ch text="新しい恋人        (第四話)"]
[if  exp="sf.aend10==1"][locate  x=20  y=195][font
color=0xFFFFFF][ch  text="新しい恋人        (第四話)
"][font color=0x666666][r][endif]¥
[locate x=20 y=210][ch text="眠っている君へ      (第四話)"]
[if  exp="sf.aend11==1"][locate  x=20  y=210][font
color=0xFFFFFF][ch  text="眠っている君へ      (第四話)
"][font color=0x666666][r][endif]¥
[locate x=20 y=225][ch text="魔導師志願        (第四話)"]
[if  exp="sf.aend12==1"][locate  x=20  y=225][font
color=0xFFFFFF][ch  text="魔導師志願        (第四話)
"][font color=0x666666][r][endif]¥
[locate x=20 y=240][ch text="白い夜明け        (第四話)"]
[if  exp="sf.aend13==1"][locate  x=20  y=240][font
color=0xFFFFFF][ch  text="白い夜明け        (第四話)
"][font color=0x666666][r][endif]¥
[locate x=20 y=255][ch text="冬嫌い         (第三話)"]
[if  exp="sf.aend14==1"][locate  x=20  y=255][font
color=0xFFFFFF][ch  text="冬嫌い         (第三話)
```

```
"][font color=0x666666][r][endif]¥
[locate x=20 y=270][ch text="消えない傷痕          （第三話）"]
[if exp="sf.aend15==1"][locate x=20 y=270][font
color=0xFFFFFF][ch text="消えない傷痕          （第三話）"][font
color=0x666666][r][endif]¥
[locate x=20 y=285][ch text="あたしのヒーロー      （第三話）"]
[if exp="sf.aend16==1"][locate x=20 y=285][font
color=0xFFFFFF][ch text="あたしのヒーロー      （第三話）"][font
color=0x666666][r][endif]¥
[locate x=20 y=300][ch text="守ってくれてありがとう（第二話）"]
[if exp="sf.aend17==1"][locate x=20 y=300][font
color=0xFFFFFF][ch text="守ってくれてありがとう（第二話）"][font
color=0x666666][r][endif]¥
[locate x=20 y=315][ch text="雪だるまになった朝    （第二話）"]
[if exp="sf.aend18==1"][locate x=20 y=315][font
color=0xFFFFFF][ch text="雪だるまになった朝    （第二話）"][font
color=0x666666][r][endif]¥
[locate x=20 y=330][ch text="来年こそは            （第一話）"]¥
[if exp="sf.aend19==1"][locate x=20 y=330][font
color=0xFFFFFF][ch text="来年こそは            （第一話）"][font
color=0x666666][r][endif]¥
[locate x=320 y=60][ch text="贖罪                  （最終話）"]
[if exp="sf.dend01==1"][locate x=320 y=60][font
color=0xFFFFFF][ch text="贖罪                  （最終話）"][font
color=0x666666][r][endif]¥
[locate x=320 y=75][ch text="人喰いベッド          （最終話）"]
[if exp="sf.dend02==1"][locate x=320 y=75][font
color=0xFFFFFF][ch text="人喰いベッド          （最終話）"][font
color=0x666666][r][endif]¥
[locate x=320 y=90][ch text="死者が扉を叩く時      （第五話）"]
[if exp="sf.dend03==1"][locate x=320 y=90][font
color=0xFFFFFF][ch text="死者が扉を叩く時      （第五話）"][font
color=0x666666][r][endif]¥
[locate x=320 y=105][ch text="山羊の仮面の呪い      （第五話）"]
[if exp="sf.dend04==1"][locate x=320 y=105][font
color=0xFFFFFF][ch text="山羊の仮面の呪い      （第五話）"][font
color=0x666666][r][endif]¥
[locate x=320 y=120][ch text="祭壇の上の君          （第四話）"]
```

```
[if exp="sf.dend05==1"][locate x=320 y=120][font
color=0xFFFFFF][ch text="祭壇の上の君          (第四話) "][font
color=0x666666][r][endif]¥
[locate x=320 y=135][ch text="偽りの夜明け          (第四話) "]
[if exp="sf.dend06==1"][locate x=320 y=135][font
color=0xFFFFFF][ch text="偽りの夜明け          (第四話) "][font
color=0x666666][r][endif]¥
[locate x=320 y=150][ch text="死者の井戸            (第三話) "]
[if exp="sf.dend07==1"][locate x=320 y=150][font
color=0xFFFFFF][ch text="死者の井戸            (第三話) "][font
color=0x666666][r][endif]¥
[locate x=320 y=165][ch text="あなたも同じ          (第二話) "]
[if exp="sf.dend08==1"][locate x=320 y=165][font
color=0xFFFFFF][ch text="あなたも同じ          (第二話) "][font
color=0x666666][r][endif]¥
[locate x=320 y=180][ch text="悪夢の続き            (第二話) "]
[if exp="sf.dend09==1"][locate x=320 y=180][font
color=0xFFFFFF][ch text="悪夢の続き            (第二話) "][font
color=0x666666][r][endif]¥
[locate x=320 y=195][ch text="あなたもオオカミに    (第二話) "]
[if exp="sf.dend10==1"][locate x=320 y=195][font
color=0xFFFFFF][ch text="あなたもオオカミに    (第二話) "][font
color=0x666666][r][endif]¥
[locate x=320 y=210][ch text="逃れ得ぬ運命          (第二話) "]
[if exp="sf.dend11==1"][locate x=320 y=210][font
color=0xFFFFFF][ch text="逃れ得ぬ運命          (第二話) "][font
color=0x666666][r][endif]¥
[locate x=320 y=225][ch text="寒さの中で            (第二話) "]
[if exp="sf.dend12==1"][locate x=320 y=225][font
color=0xFFFFFF][ch text="寒さの中で            (第二話) "][font
color=0x666666][r][endif]¥
[locate x=320 y=240][ch text="ついてきたもの        (第一話) "]
[if exp="sf.dend13==1"][locate x=320 y=240][font
color=0xFFFFFF][ch text="ついてきたもの        (第一話) "][font
color=0x666666][r][endif]¥
[locate x=320 y=255][ch text="痴話喧嘩              (序　章) "]
[if exp="sf.dend14==1"][locate x=320 y=255][font
color=0xFFFFFF][ch text="痴話喧嘩              (序　章) "][font
```

```
color=0x666666][r][endif]¥
[locate x=320 y=285][ch text="各種一発死                （第七話）"]
[if  exp="sf.dend15==1"][locate  x=320  y=285][font
color=0xFFFFFF][ch  text="各種一発死              （第七話）"][font
color=0x666666][r][endif]¥
[locate x=320 y=300][ch text="各種一発死                （第六話）"]
[if  exp="sf.dend16==1"][locate  x=320  y=300][font
color=0xFFFFFF][ch  text="各種一発死              （第六話）"][font
color=0x666666][r][endif]¥
[locate x=320 y=315][ch text="各種一発死                （第五話）"]
[if  exp="sf.dend17==1"][locate  x=320  y=315][font
color=0xFFFFFF][ch  text="各種一発死              （第五話）"][font
color=0x666666][r][endif]¥
[locate x=320 y=330][ch text="各種一発死                （第三話）"]¥
[if  exp="sf.dend18==1"][locate  x=320  y=330][font
color=0xFFFFFF][ch  text="各種一発死              （第三話）"][font
color=0x666666][r][endif]¥
[resetfont]¥
[locate  x=440  y=400][link  target=*finish_memories
color=0xFF0000]メニューに戻る[endlink]¥
[backlay]¥
[layopt layer=message0 page=back visible=true]¥
[scenechange4 storage="title2"]¥
[current layer=message0]¥
[s]
;-----------------------------------------------
*finish_memories
[fadeoutbgm time=1000]¥
[backtxtclear]¥
[scenechange storage="title2"]¥
[er]¥
[wb]¥
[jump target=*option2]
[s]
```

9-4 制限時間つき選択肢を作る

市販ゲームの中には選択肢に制限時間が設けられているものがありますが、KAGでは『制限時間つき選択肢』という概念はなく、[wait]タグと[jump]タグを組み合わせることで非常に簡単に実現できます。

●シナリオ例

```
[nowait]¥
[locate y=365]¥
[link target=*scene01]選択肢A[endlink]
[link target=*scene02]選択肢B[endlink]
[link target=*scene03]選択肢C[endlink]¥
[endnowait]¥
[wait time=制限時間]¥
[jump target=*時間切れのページへ]¥
[s]
```

今までは選択肢の終わりには[s]タグをつけて強制的に動きを止めていましたが、選択肢の次に[wait]タグを置くことによって、ここでスクリプトの処理を時間制限をつけて停止させることができます。この[wait]で指定された時間（単位ミリ秒）を過ぎるとスクリプトは下の行にある[jump]タグを実行します。これで、制限時間つきの選択肢が出来ました。

ただしこの例では、何秒待ったら時間切れになるのか分かりません。そこで、前景レイヤーにメーターのような画像を表示したユニバーサル・トランジションを利用します。画像を準備さえすれば、円形やバーメーターのようないろいろな形で残時間が表現できます。

●準備するファイル

残時間のあるメーター画像ファイル	limit01.png
残時間のないメーター画像ファイル	limit02.png
トランジションのためのルール・ファイル	meter.png

●シナリオ例

```
1    [layopt layer=0 page=fore visible=true]¥
2    [image storage="limit01.png" layer=0 page=fore]¥
3    [nowait]¥
4    [history output=false]¥
5    [locate y=300]¥
6    [link target=*s01]選択肢A[endlink]
7    [link target=*s02]選択肢B[endlink]
8    [endnowait]¥
9    [backlay]¥
10   [cancelskip]¥
11   [clickskip enabled=false]¥
12   [image storage="limit02.png" layer=0 page=back]¥
13   [trans rule="meter.png" time=10000 vague=10]¥
14   [wt]¥
15   [jump target=*timeout]¥
16   ;------------------------
17   *s01
18   [stoptrans]¥
19   [freeimage layer=0]¥
20   [clickskip enabled=true]¥
21   [ct]¥
22     選択肢Aが選択されました。[l]
23   ;------------------------
24   *s02
25   [stoptrans]¥
26   [freeimage layer=0]¥
27   [clickskip enabled=true]¥
28   [ct]¥
29     選択肢Bが選択されました。[l]
30   ;------------------------
31   *timeout
32   [ct]¥
33   [freeimage layer=0]¥
```

＊編注
実際のシナリオには、行番号はいりません。

```
34   [clickskip enabled=true]¥
35   [s]
```

≪1～2行目≫

まずは前景レイヤー0のフォアグラウンドを可視状態にし、そこに「limit01.png」というメーター用画像ファイルを表示し、続いてメッセージ・レイヤー0に選択肢を表示します。

≪10～11行目≫

続いて「limit02.png」という残時間の存在しない画像ファイルを前景レイヤー1の裏画面に読み込み、「meter.png」というルール・ファイルで「limit01.png」から「limit02.png」へトランジションを行なっています。

時間内に選択肢を選んだ場合はジャンプ先のラベルで[stoptrans]というタグを用い、強制的にトランジションを停止します。これを行なわないと選択肢を選んだのにメーター・ファイルが稼動し続けることになり、おかしなことになります。

≪26、33行目≫

また、[freeimage]というタグは、指定された前景レイヤー0の内容を強制的に解放します。この場合はメーター関係の画像データをクリアして前景レイヤー0を空にしています。このように使い終えたらきちんと元の状態に戻しておくことで、余分なメモリを消費することもなくなり、次に前景レイヤー0を利用しようとした時に前のデータが残っていることもなくなります。

メーター・ファイルと同時に背景画像も表示する場合は、メーター・ファイルを透明度情報を持つ画像で作らないといけません。

アンチエイリアスが掛かっていないシンプルな形状のメーターを利用する場合は、[image]タグのkey属性を利用します。

key属性は、透明にしたい色を「key=0xRRGGBB」で指定すると、その色が透明になる効果を持ちます。「key=adapt」と指定すると、画像の最上部のラインで最も多く使われている色ピクセルの色が自動的に抜き色となります。

●記述例

```
[image　storage="meter.png"　layer=0　page=fore
key=0x000000]¥
```

ただし、上の方法でアンチエイリアスが掛かっている画像の背景色を抜こうとすると縁が汚くなります。このような場合は、「吉里吉里２」に付属する画像フォーマット・コンバータを使うといいでしょう。

画像フォーマット・コンバータは「kr2_218r2¥kirikiri2¥graphconv」の中に入っている「krkrtpc.exe」です。

まず、PhotoShopのPSDファイルなどの透過背景を持つ画像を準備し、それを画像フォーマット・コンバータにドロップするだけで、KAGの前景レイヤーに読み込むことのできる画像が生成されます。

通常利用する場合、「透過情報を持った画像の出力形式(F)」の部分は「αチャンネル付PNG（メイン＋マスク）」で構いませんが、CDでの配布を前提としており、サイズは小さくなくてよいから展開速度を上げたいという場合は32bitBMP（メイン＋マスク）を選びましょう。

9-5 ムービーを再生する[video][wv]

*Flash
Macromedia社が開発したWeb用アニメーション・コンテンツを作るソフト。

*Premiere
Adobe社が開発したデジタル・ムービー編集ソフト。

Flash*やPremiere*などのソフトが使える人は、オープニングに動画やアニメーションなどのムービーを入れたいと思うでしょう。

KAGでは専用のタグが用意されており、MPEG形式のムービーとSWF形式のFlashファイルを再生することができます。

●シナリオ例

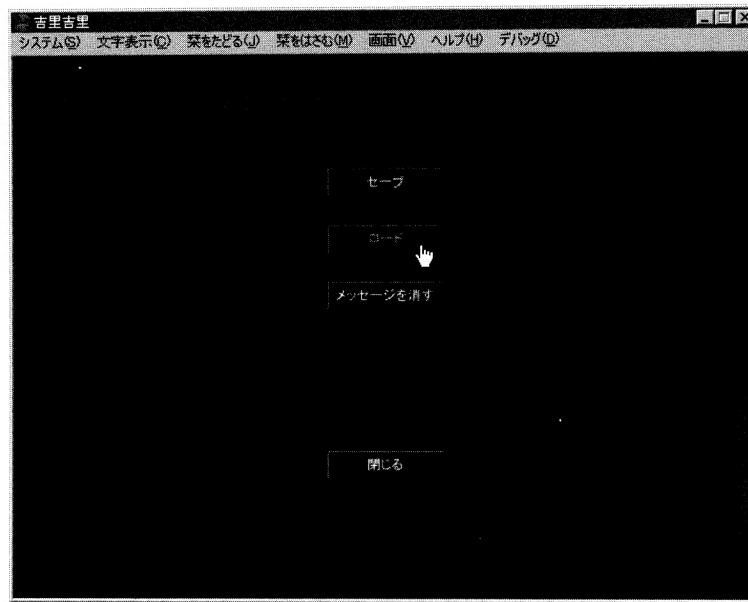
```
*start
[video visible=true left=0 top=0 width=640 height=480]¥
[playvideo storage="logo.mpg"]¥
[playvideo storage="Love.swf"]¥
[wv]¥
```

「krkr.eXe」と同じ場所にFlashファイルを再生するには「krkr.eXe」と同じ場所に「krflash.dll」プラグインを、MPEGなどの動画を再生するには「krmovie.dll」プラグインを置く必要があります。また、「first.ks」の冒頭などでプラグインをロードする必要があります。

●記述例

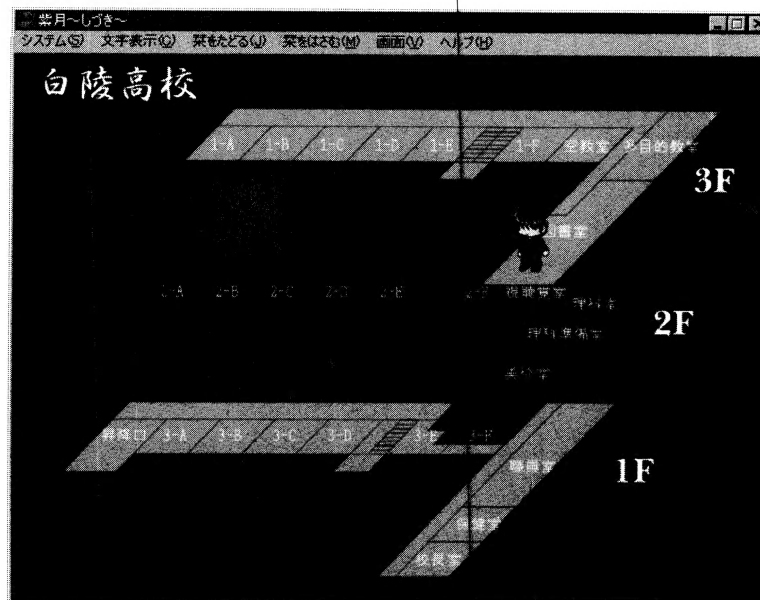
```
[loadplugin module="krmovie.dll"]¥
[loadplugin module="krflash.dll"]¥
```

9-6 右クリック動作のカスタマイズ

「吉里吉里/KAG」のデフォルトの機能では、マウスを右クリックした場合メッセージ・レイヤーが非表示になります。しかし、この右クリックで固定のサブルーチンを呼ぶ動作を定義することができます。

画面例１

これはKAG3のサンプルとして付属しているプラグインの一つである「¥kag3¥kag3plugin¥sample¥」内の「rclick_tjs.ks」と「rclick_tjs_test.ks」を使ってみた結果、市販のゲームのように、右クリックで表示されるメニューにおいてゲームの進行データをセーブしたりロードしたりできるようになった例です。

やり方ですが、「rclick_tjs_test.ks」内に記述されているサンプル・シナリオの必要な部分を「first.ks」にコピーし、「rclick_tjs.ks」ファイルをscenarioフォルダにそのままコピーするだけです。

「rclick_tjs_test.ks」内に記述されていた「@call storage="rclick_tjs.ks" target=*rclick」の部分で「rclick_tjs.ks」という外部サブルーチン・ファイルを呼び出し、上のような右クリックメニューを実現しています。

画面例 2

サンプルの『紫月』に搭載されている右クリックによるマップ表示です。

これも同様に、メインシナリオ側から「@call storage="rclick.ks" target=*rclick」という記述で「rclick.ks」という名前のサブルーチン・ファイルを呼び出しています。

また、もっとも基本的な右クリック・メニューを作るための「rclick.ks」はKAGシステム・リファレンス内に記述されているのでご覧ください。この場合、ラベルが「*sub1」なので、first.ksなどのメインシナリオからは「@call storage="rclick.ks" target=*sub1」というように記述して呼び出します。

9-7 zoomプラグインの利用

背景や前景に表示した画像を拡大したり縮小したりするには、「zoomプラグイン」を使います。このプラグインは「KAG3¥KAG3plugin¥zoom¥scenario¥」の中にある「zoom.ks」が本体です。まずは、これを自分のプロジェクト・フォルダ下部のscenarioフォルダにコピーしておきましょう。

続いて、シナリオの書き方ですが、zoomプラグインを使いたいプロジェクトの「first.ks」の先頭行に、以下のように記述します。

●記述例

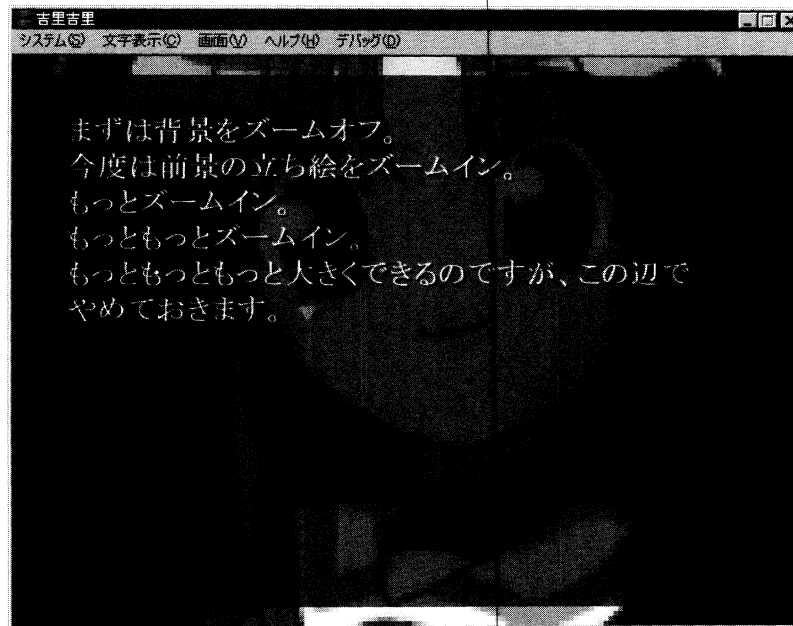
```
@call storage="zoom.ks"
```

これも前述の「staffroll.ks」と同様、サブルーチン形式で提供されており、このサブルーチンを呼び出すことで、以下の4つのzoom用タグが定義されます。

bgzoom	baseレイヤーの画像をズームする
fgzoom	前景レイヤーの画像をズームする
wbgzoom	baseレイヤーの画像のズーム終了を待つ
wfgzoom	前景レイヤーの画像のズーム終了を待つ

●シナリオ例

```
[image storage="test01.bmp" layer=base page=fore]¥
まずは背景をズームオフ。[l]
[image storage="black.bmp" layer=base page=fore]¥
[bgzoom storage="test01.bmp" bgstorage="test01.bmp" sl=0
st=0 sw=640 sh=480 dl=320 dt=240 dw=0 dh=0 time=3000
sccel=0]¥
[wbgzoom]¥
今度は前景の立ち絵をズームイン。[l]
[laycount layers=1]¥
[fgzoom storage="akira.png" layer=0 mode=transp sl=320
st=480 sw=0 sh=0 dl=160 dt=0 dw=320 dh=480 time=3000
sccel=0]¥
```

```
[wfgzoom]¥
もっとズームイン。[l]
[fgzoom storage="akira.png" layer=0 mode=transp sl=160
st=0 sw=320 sh=480 dl=0 dt=-240 dw=640 dh=960 time=1000
sccel=0]¥
[wfgzoom]¥
もっともっとズームイン。[l]
[fgzoom storage="akira.png" layer=0 mode=transp sl=0 st=-
240 sw=640 sh=960 dl=-320 dt=-720 dw=1280 dh=1920
time=1000 sccel=0]¥
[wfgzoom]¥
もっともっともっと大きくできるのですが、この辺でやめておきます。[l]
```

画像のズーム

9-8 雪・雨プラグインの利用

「KAG3」には、前景などに雨や雪を降らせるプラグインが準備されています。これらのプラグインは「KAG3¥KAG3plugin¥」内にある「rain」と「snow」フォルダに格納されているので、利用する場合はこれらのプラグインを自分のプロジェクト・フォルダにコピーしておきます。

各プラグインフォルダ内のimageフォルダに含まれている画像は「fgimage」や「image」「other」などのフォルダへ、そしてscenarioフォルダ内に含まれているksファイルはscenarioフォルダにコピーします。

「staffroll」プラグインや「zoom」プラグインと同様、まずはプロジェクトの「first.ks」の先頭行にプラグインをロードするためのサブルーチン呼び出しコマンドを記述します。

ワンポイント
画像のコピーをし忘れるとプラグインは動作しません。雪と雨のプラグインについては必ずシナリオ・ファイルと画像ファイルの両方をプロジェクト・フォルダにコピーしましょう。

●記述例

```
@call storage="snow.ks"
@call storage="rain.ks"
```

雪と雨の両方のプラグインを使いたければ、上の記述例のように両方記述する必要があります。

●シナリオ例

```
[image storage="test02.bmp" layer=base page=fore]¥
では、雪を降らせます。[l]
@snowinit forevisible=true backvisible=false
しばらく待っていると上のほうから雪が降ってくると思います。[l]
細かい調整はsnow.ksを変更することで可能になります。[l]
では止めてみましょう。[l]
@snowuninit
止みました。[l]
では、雨を降らせます。[l]
@raininit forevisible=true backvisible=false
どうですか？　雨が降っていますか？[l]
雨の振り方はrain.ksを調整することで変更が可能です。[l]
では止めてみましょう。[l]
@rainuninit
止みました。[l]
```

Step 10 リリースに向けて

各種カスタマイズの方法や、バージョン情報ダイアログの作り方など、作品が完成に近づいたら理解しておきたい機能を中心に解説します。

■セーブ・データのロード画面

ここでは、一般のゲームなどでメニューから「ロード」を選ぶと表示される専用のデータロード画面を作ってみます。利用するタグは選択肢を配置する時に使った[link]タグと、変数を表示する[emb]タグの２つです。

●シナリオ例

```
[link exp="kag.restoreBookMark(0)"]¥
[emb exp="kag.getBookMarkPageName(0)"]¥
[endlink]¥
```

[emb]タグで「PageName(0)」という栞のセーブ・タイトルを表示し、それを[link][endlink]でくくり、クリックされたら「BookMark(0)」というセーブ・データをロードするという意味です。

KAGの栞データには「0」から「998」までの番号が与えられており、１番目の栞には「0」、２番目の栞には「1」、３番目は「2」、４番目は「3」というように番号が振られています。このため上記の例は「１番目のセーブデータを読み込む」という意味になります。

これを元にしてロード画面を作ると、以下のようになります。

●シナリオ例

```
*dataload
[ct]¥
[nowait]¥
[locate x=40 y=20]再開する栞を選んでください。
[style align=center]¥
[link              exp="kag.restoreBookMark(0)"][emb
exp="kag.getBookMarkPageName(0)"][endlink]
[link              exp="kag.restoreBookMark(1)"][emb
exp="kag.getBookMarkPageName(1)"][endlink]
[link              exp="kag.restoreBookMark(2)"][emb
exp="kag.getBookMarkPageName(2)"][endlink]
[link              exp="kag.restoreBookMark(3)"][emb
```

```
exp="kag.getBookMarkPageName(3)"][endlink]
[link               exp="kag.restoreBookMark(4)"][emb
exp="kag.getBookMarkPageName(4)"][endlink]
[style align=default]¥
[locate x=400 y=360][link target=*menu]メニューに戻る
[endlink]¥
[endnowait]¥
[s]
```

[link]タグで表示される矩形の色を変えたい場合は、以下のように記述します。

```
[link exp="kag.restoreBookMark(0)" color=0xff0000]¥
```

このcolor属性は[link]タグを使ったすべての選択肢に使えるので、工夫をすれば面白い選択肢を作ることができるでしょう。

■終了処理

ゲームを終了させたい場合の選択肢は、下記のように書きます。

```
[link exp="kag.close()"]ゲームを終了する[endlink]¥
```

色をつける場合は、以下のように記述します。

```
[link exp="kag.close()" color=0xff0000]ゲームを終了する
[endlink]¥
```

■改行／改ページ待ち記号のカスタマイズ

緑や青の三角で表わされる改行待ち記号をカスタマイズする方法を紹介します。[l](小文字のエル)タグを書いたときに表示される改行待ち記号は、以下のファイルから成り立っています。

```
LineBreak.png
LineBreak_a.png
LineBreak.asd
```

また、[p]タグを書いたときに表示される改ページ待ち記号は、以下のファイルから成り立っています。

```
PageBreak.png
PageBreak_a.png
PageBreak.asd
```

これは実は、前景レイヤーのアニメーション時に説明したアニメーション・ファイルとまったく同じ構造をしています。「ファイル名.png」がアニメーションの元となるベース・ファイル、「ファイル名_a.png」がアニメーション・ファイル、そして「ファイル名.asd」がアニメーションの動作を記述するファイルです。

これらの記号の色を変えるには、「PageBreak_a.png」や「LineBreak_a.png」などのアニメーション用画像ファイルを画像処理ソフトで開き、色相を変えるだけです。ベース・ファイル上に描かれている16個の記号をそれぞれ選択し、色を塗ってもかまいません。

また、形を変える場合は、「PageBreak_a.png」や「LineBreak_a.png」を書き直す必要があります。

この2つのファイルは、24×24ドットの小さな画像が横に16個並べられた16コマのアニメーションになっているので、24×24のサイズの画像を16個作って、それを横に並べれば、オリジナルの記号が作れます。

●LineBreak_a.png例

●PageBreak_a.png例

■マウス・カーソルのカスタマイズ

吉里吉里で制作した作品では、オリジナルのマウス・カーソルが使えます。オリジナルのカーソルには「アニメーション・カーソル」(.ani) と「一般のカーソル」(.cur)の2種類が指定できます。

まずは拡張子が「cur」か「ani」のファイルを作ります。WindowsのCursorsフォルダ内にたくさん入っているので、初めての人はこちらから適当なものを選ぶとよいでしょう。自分で作りたい人は専用のソフトを使うか、インターネットで配布されているフリーのマウスカーソル・エディタなどを手に入れて、それで作ります。

curファイルが手に入ったら、それをtemplateフォルダ内のotherフォルダにコピーします。

続いて、「Config.tjs」の以下の部分を開きます。

```
// ◆ マウスカーソル
// マウスカーソルを指定します。マウスカーソルは cr で始まるマウスカ
// ーソル定 数 ( 吉里吉里2 SDK Help 参照 ) か、マウスカーソルフ
//  ァイル名である必要があります。アニメーションマウスカーソルも指定
// できます。
// マウスカーソル定数を指定するときは、cursor タグで指定するように
```

```
// 先頭に&をつける必要はありません。マウスカーソルファイル名を指定
// する場合は ""(ダブルクオーテーション) でくくってください。
;cursorDefault = crArrow; // 通常のマウスカーソル
;cursorPointed = crHandPoint; // リンクなどをポイントしたとき
;cursorWaitingClick = crArrow; // クリック待ちの時
;cursorDraggable = crSizeAll; // メッセージレイヤをドラッグ可
                                 能なとき
```

下部の4行を見れば、吉里吉里で利用されているマウス・カーソルは4種類あることが分かりますが、通常時とクリック待ちのときに同じファイルが指定されているので、実質的には3種類になります。なぜ通常時とクリック待ちのときに同じカーソルを指定しているのかというと、この2つの状態はお互いに小刻みに入れ替わることが多く、別のカーソルを指定するとチラチラしてしまうからです。

この部分に、以下の記述例のようにオリジナルのカーソル名を指定します。ファイル名は拡張子まで指定し、「""」(ダブル・クオーテーション) でくくっておくと、間違いがありません。

```
;cursorDefault ="original01.cur"; // 通常のマウスカーソル
;cursorPointed ="original02.cur"; // リンクなどをポイントしたとき
;cursorWaitingClick ="original01.cur"; // クリック待ちの時
;cursorDraggable =original03.cur"; // メッセージレイヤをドラ
                                      ッグ可能なとき
```

これでオリジナルのカーソルが使えますが、これとは別にゲーム・シナリオの中で一時的にカーソルを変更したい場合は[cursor]タグを用います。

●タグ記述例

```
[cursor default="original01.cur" pointed="original02.cur"
click="original01.cur" draggable="original03.cur"]¥
```

[cursor]タグの属性は、それぞれが「Config.tjs」の4行部分に該当します。

■ダイアログのカスタマイズ

栞を挟んだり、たどったり、ゲームを終了したりするときに表示される「はい／いいえ」ダイアログもカスタマイズすることができます。

まずは、基本となる、どのようなダイアログでも常に表示される文字部分を変更してみます。

「systemフォルダ」内の「YesNoDialog.tjs」を開き、以下の部分を探します。

●YesNoDialog.tjs例

```
// Yesボタン
add(yesButton = new ButtonLayer(this, primaryLayer));
yesButton.caption = "はい";
yesButton.captionColor = clBtnText;
yesButton.width = 70;
yesButton.height = 25;
yesButton.top = th + 35;
yesButton.left = (w - (70*2 + 10)>>1);
yesButton.visible = true;

// Noボタン
add(noButton = new ButtonLayer(this, primaryLayer));
noButton.caption = "いいえ";
noButton.captionColor = clBtnText;
noButton.width = 70;
noButton.height = 25;
noButton.top = th + 35;
noButton.left = ((w - (70*2 + 10))>>1) + 70 + 10;
noButton.visible = true;
```

ここに記述されている「はい」と「いいえ」がダイアログに表示される文字なので、自分の好きなように修正します。ダイアログで「確認」と表示されている部分は、「YesNoDialog.tjs」の最下部にあります。

●Menu.tjs例

```
// Yes か No かはっきりさせる関数
function askYesNo(message, caption = "確認")
{
	var win = new YesNoDialogWindow(message, caption);
	win.showModal();
	var res = win.result;
	invalidate win;
	return res;
}
```

この「確認」という部分を書き換えれば、変更が適用されます。

また、これらの基本的な文字以外（たとえば「終了しますか？」などのメッセージ）はMainWindow.tjs内で定義されています。

たとえば、「終了しますか？」というメッセージを変更したい場合は、MainWindow.tjs内の以下の部分を変更します。

●MainWindow.tjs例

```
//-----------------------------------onCloseQuery/close
function onCloseQuery()
{
	saveSystemVariables();
	if(!askOnClose)  { super.onCloseQuery(true); return;
}
	super.onCloseQuery(askYesNo("終了しますか？"));
}
```

上記の「終了しますか？」の部分を書き換えれば、変更した文字がダイアログに反映されます。

■メニューのカスタマイズ

メニュー・バーに表示されている「システム／栞をたどる／栞をはさむ／画面」のような親メニュー、そしてそれらをクリックすると表示される子メニューについては自由にカスタマイズできます。

セーブロードを使わない作品の場合、『栞』関係のメニューは表示しておいても邪魔なだけなので、「false」にしてメニュー・バーから消しておきましょう。

また、会話がほとんど存在せず、パラメータのみで構成されている「育成シミュレーションゲーム」のような場合には、メッセージ履歴を表示させても仕方がないので、消しておきます。

選択肢のない短編の電子小説の場合なら、『次の選択肢／未読まで進む』というメニューは不要なので消しましょう。

これらは、次の「Confit.tjs」内の記述を変更することで実現できます。

●Config.tjs例

```
//--------------------------------------------メニューの設定
function Menu_visible_config()
{
// メニューの表示/非表示の設定
// true を指定すると表示され、false を指定すると非表示になります。
// メニューに表示するメニュー項目名については Menus.tjs を書き換
// えてください。
// ◆ メニューバーを表示するか
// 非表示にすると当然どのメニュー項目にもアクセスできなくなります。
;menu.visible = true;
// ◆ 「システム > メッセージを消す」
// このメニュー項目を非表示にしてもマウスの右クリックでのメッセージ
// の非表示は引き続き有効です。

;rightClickMenuItem.visible = true;
↑背景だけを見せる必要がなければfalseに。
// ◆ 「システム > メッセージ履歴の表示」
;showHistoryMenuItem.visible = true;
```

```
↑メッセージ履歴を使う必要がなければfalseに。
// ◆ 「システム > 次の選択肢/未読まで進む」または「次の選択肢まで
// 進む」
;skipToNextStopMenuItem.visible = true;
↑選択肢がない作品の場合はfalseに。
// ◆ 「システム > 自動的に読み進む」
;autoModeMenuItem.visible = true;
// ◆ 「システム > 自動的に読み進むウェイト」
;autoModeWaitMenu.visible = true;
// ◆ 「システム > 前に戻る」
;goBackMenuItem.visible = true;
↑選択肢がない作品の場合はfalseに。
// ◆ 「システム > 最初に戻る」
;goToStartMenuItem.visible = true;
↑最初に戻る必要がなければfalseに。
// ◆ 「文字表示」
;characterMenu.visible = true;
// ◆ 「文字表示 > 表示速度 > ページ末まで一気に」
// ユーザがこのメニュー項目をチェックすると、
// l タグが無視されるようになります。
;chNonStopToPageBreakItem.visible = true;
// ◆ 「文字表示 > 一度読んだところは」
;ch2ndSpeedMenu.visible = true;
// ◆ 「文字表示 > 一度読んだところは > ページ末まで一気に」
;ch2ndNonStopToPageBreakItem.visible = true;
// ◆ 「文字表示 > アンチエイリアス」
;chAntialiasMenuItem.visible = true;
// ◆ 「文字表示 > フォント」
;chChangeFontMenuItem.visible = true;
↑フォントをユーザーに変更させたくなければfalseに。
// ◆ 「栞をたどる」
;restoreMenu.visible = true;
↑栞を使わない作品の場合はfalseに。
// ◆ 「栞をはさむ」
```

```
;storeMenu.visible = true;
↑栞を使わない作品の場合はfalseに。
// ◆ 「画面」
;displayMenu.visible = true;
↑画面の表示サイズを変更させたくなければfalseに。
// ◆ 「ヘルプ」
;helpMenu.visible = true;
↑ヘルプがない作品の場合はfalseに。
// ◆ 「ヘルプ > 目次 ...」
;helpIndexMenuItem.visible = false;
// ◆ 「ヘルプ > このソフトについて ...」
;helpAboutMenuItem.visible = false;
// ◆ 「デバッグ」
;debugMenu.visible = true;
↑リリース前に必ずfalseに戻しておく。
```

「Confit.tjs」を書き換えた場合は、「吉里吉里」を再起動しないと変更が適用されないので、注意してください。

また、systemフォルダ内の「Menu.tjs」を直接書き換えることによってメニューに表示されている言葉を変更できます。

たとえば、

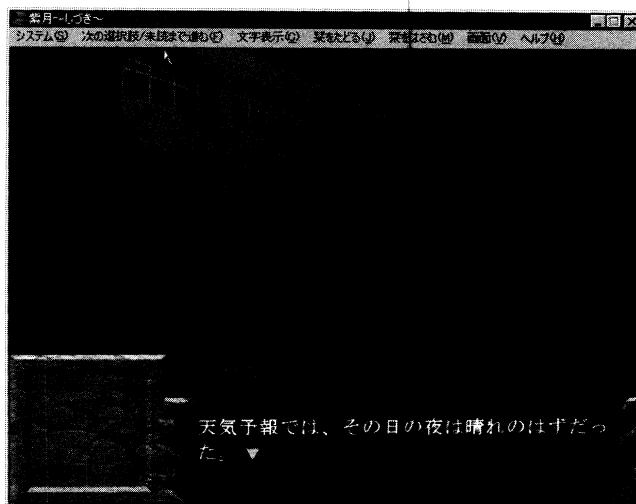
```
menu.add(this.systemMenu = new KAGMenuItem(this, "システム
(&S)", 0, "", false));
```

の行の「システム」という部分を「環境設定」のように変更すれば、「吉里吉里」のメニューに反映されます。

メニュー・コマンドを直接メニュー・バーに表示したい場合は、「MainWindow.tjs」を書き換えなくてもTJSスクリプトを書けば可能です。「first.ks」の先頭などに、以下のような記述をします。

●シナリオ例

```
[iscript]
kag.systemMenu.remove(kag.skipToNextStopMenuItem);
kag.menu.insert(kag.skipToNextStopMenuItem, 1);
[endscript]
```

2行目で「次の未読／選択肢まで進む」というメニュー・コマンドを「システム」の子の部分から外し、**3行目**で直接メニュー・バーに表示しています。上の例では、「次の未読／選択肢まで進む」を変更するため「kag.skipToNextStopMenuItem」という定義名を使っていますが、各メニュー・コマンドがどのような名前で定義されているかは「Menu.tjs」を見て研究してください。

■バージョン情報ダイアログの作成

「このソフトについて」のような情報ダイアログを作るには、Config.tjsの該当部分を以下のように設定します。

```
// ◆ 「ヘルプ > 目次 ...」
;helpIndexMenuItem.visible = true;

// ◆ 「ヘルプ > このソフトについて ...」
;helpAboutMenuItem.visible = true;
```

```
// ◆ 「ヘルプ > このソフトについて」を選択したときにでる
// ウィンドウの表示領域のサイズ
// このメニューを選択すると about.ks がそのウィンドウ内で実行され
// ます。
;aboutWidth = 320; // 幅
;aboutHeight = 200; // 高さ
```

上の3つの部分の設定が終了したら、「about.ks」というファイルを作ります。この「about.ks」は通常のシナリオとほとんど一緒ですが、いくつか制限があります。

・効果音バッファは1つしか使えない
・ムービーは再生できない
・メッセージ・レイヤーは1つしか使えない
・メッセージ履歴は表示できない

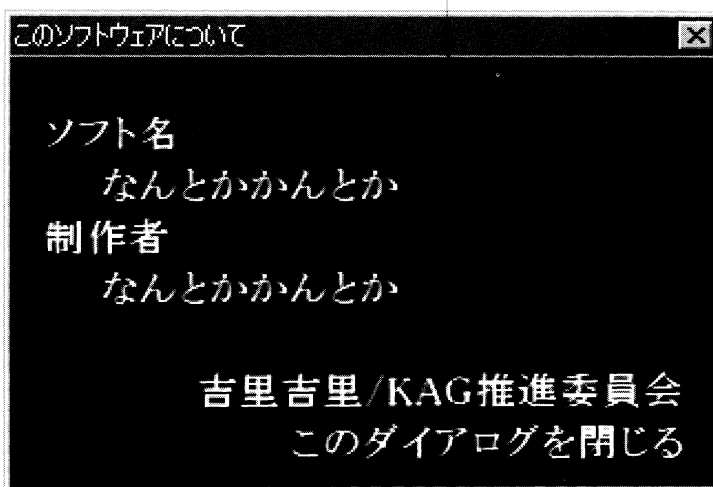
上記の点を踏まえて、通常のシナリオを書くような感じで書きましょう。

●シナリオ例

```
*set
[title name="このソフトウェアについて"]¥
[image storage="help.bmp" layer=base page=fore]¥
[position left=0 top=0 width=320 height=200
color=0xffffff opacity=0 marginl=15 margint=15 marginr=15
marginb=15]¥
[nowait]¥
[font size=18]¥
ソフト名
    なんとかかんとか
制作者
    なんとかかんとか
[style align=right]
[link                                   color=0xff0000
```

```
exp="System.shellExecute('http://www.piass.com/kpc/')"
hint="吉里吉里/KAGの公式サポートサイト"]吉里吉里/KAG推進委員会
[endlink]
[link target=*exit]このダイアログを閉じる[endlink]¥
[endnowait]¥
[s]
*exit
@close
```

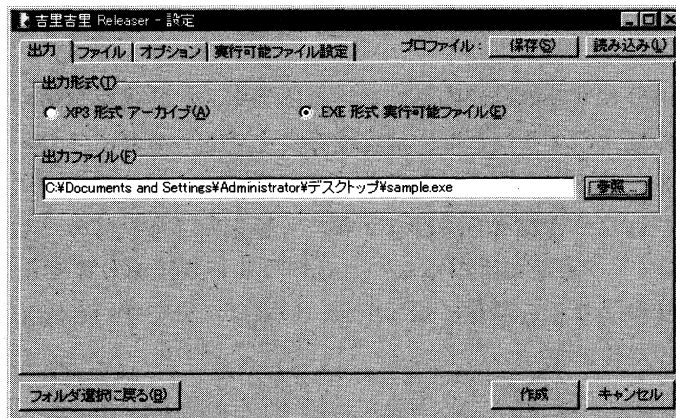
上記の[link]タグの例のようにexp属性で「System.shellExecute」にURLを指定しておくと、リンクをクリックしたらブラウザを起動して指定のサイトにアクセスするようになります。

この例は「about.ks」内だけでなく、一般のシナリオ内でも利用できるので、クリックしたときに指定のホームページにアクセスさせたい場合は上記のように書くとよいでしょう。ただし、ユーザーの誰もがインターネットへ常時接続できる環境を持っているとは限らないので、多用は禁物です。

■Releaserによる実行ファイル化

細かい部分を詰め終わり、プロジェクトが完成したらReleaserを用いてリリースします。「リリース」は、直訳すると『公開する／発売する』という意味ですが、『解放する／手放す』という意味もあります。

KAGで制作したプロジェクトは、専用の「リリーサー」を使ってリリースします。この作業をすることによって、画像やサウンド、シナリオなどのファイルがバラバラにフォルダの中に入っていた状態を、大きな一つのファイルにまとめることができます。

この作業を行なうことで、市販のゲームのように、プログラムである実行ファイルが1個、セーブ・データを保存するフォルダが1個、そしてドキュメントが1個という、非常にシンプルでカッコいい形態で配布することができるわけです。

リリーサーを起動したら、リリースしたいプロジェクト・フォルダを指定し、アイコンをオリジナルのものに変更すれば、必要最小限の設定は完了です。すべての設定が終わったら作成ボタンを押せば、exe形式にリリースされます。

Releaserの設定

ワンポイント

readme.txtは昔は「readme.doc」という名前のファイルでやりとりされていました。ところがMicrosoft-Wordというワープロソフトの標準拡張子が「doc」に設定されてしまったため、readme.docファイルをダブルクリックすると重いWordがいちいち開くことになってしまい、ひどく迷惑なことになりました。このためWordの登場後は、テキストファイルのdoc拡張子は影をひそめ、readmeもtxt拡張子を使うようになったのです。

■配布における注意点

圧縮ファイルにして配布するときは、リリーサーで生成した実行形式ファイルの他にreadme.txtを書きましょう。

このファイルは、作品の説明や利用条件、バージョンアップ情報などを記載した『わたしを読んでね』という名前のテキスト・ファイルのことで、パソコン通信上でMS-DOS時代のプログラムがやり取りされていたころから存在しています。

readmeファイルは、以下の三種類の形式で作ることができます。

- 純然たるテキスト・ファイル
- html形式
- Windowsのヘルプファイル形式

Windows標準のヘルプ・ファイル形式である「hlp」形式は、作るにはコツが必要ですし、一部利用しにくい点があるため、あまり使わないほうがよいでしょう。これが持つメリットの多くは馴染み深いhtml形式が兼ね備えているのでhtmlで作ったほうがよいかもしれません。

ただし、htmlにするほどの内容でなければテキスト形式をお勧めします。テキスト形式の最大のメリットは、対応するテキスト・エディタやメモ帳が短時間で起動でき、すぐに内容を読むことができる点です。このように、ゲームの説明を簡単に行なう程度なら純粋なテキスト形式のほうがベターです。

readmeは「こう書かねばならない」というルールはありません。すでに配布されている作品などを参考にしながら、いちばん見やすいと思えるものを真似するとよいでしょう。

こうして、実行ファイルとreadme.txtが準備できたら、圧縮ユーティリティを使ってLZH形式などの圧縮ファイルに圧縮します。「Config.tjs」内で指定した、ゲームのセーブ・データを保存するフォルダについては、「吉里吉里２」では自動的に作成されるようになったので、同梱する必要はありません。

ただし、CD-ROMなどの書き込み不可能なメディア上で実行ファイルを実行すると、savedataフォルダがCD-ROM上に作成できないため、エラーになります。回避するには、インストーラを用いてハードディスクにインストールするか、実行ファイルを手動でハードディスク上にコピーしてもらうようreadmeで説明をする必要があります。

第1章 TJSとは

ここではある程度KAGを使ったことがあり、コンピュータに関する知識はあるがTJSがどういうものかを知らないという方を対象に、KAGをサンプルにしながらTJSの解説をします。

1-1 KAGとTJS

ほとんどの方は「吉里吉里」というと「KAG」を使っていると思います。KAGを使えば、「TJS」がどういうものであるかを知らなくても、吉里吉里上でプログラミングができます。そのため、「TJS」がいったいどういうもので何をやっているかを知る機会は少ないかと思います。

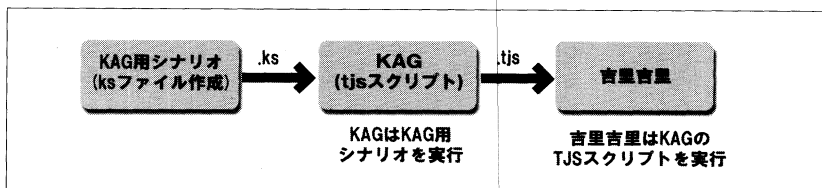
簡単に言うと、TJSは、「吉里吉里が直接理解できるスクリプト言語*」です。逆に言えば、吉里吉里はTJS以外の言語を理解できません。

*スクリプト言語
script language
アプリケーションの機能を補完する簡易言語。Perlなどもこれに含まれる。

でも、「KAGでもプログラミングできるよ?」「あれ?KAGで今まで書いていたのは吉里吉里用のスクリプトじゃないの?」ということになりますよね。違うんです。

ここではっきりさせておきましょう。

・「KAG」は「KAG用のシナリオ」を実行する。
・「吉里吉里」はその「KAG」を実行する。
・「KAG」は「TJS」で書かれている。

ということなのです。

吉里吉里はKAG用のシナリオ・ファイルを直接実行しているわけではないのです。KAGは「吉里吉里をノベルアドベンチャーゲーム・エンジンとして動作させるためのスクリプト」という、ちょっとややこしい表現がされているのはこのためです。吉里吉里がノベルアドベンチャーゲーム・エンジンとして動作するためのロジックがすべてTJSで書かれているのです。

■ KAGを読もう

KAGを使っている皆さんは、KAGのsystemフォルダを必ず一度は覗いたことがあるでしょう。さらに、一度はそのフォルダにある「Config.tjs」を編集したことがあると思います。

「Config.tjs」の拡張子「tjs」はいったい何を表わしているのでしょうか。

そうです。「TJSって何だろう」と思いつつ、必要に駆られていじっていたこの「Config.tjs」…それこそがまさに「tjsスクリプト」なのです。

他にも拡張子が「tjs」になっているファイルがいくつもあると思いますが、これらが全部TJSスクリプト・ファイルです。KAGがどのように動くべきかがすべて記述されたファイルたちなのです。

TJSを学ぶ場合、いったいどこから入門したらよいか迷うと思います。入門の「門」はさまざまなところにあると思いますが、ここではKAGを読み砕くことによってTJSを学んでいくことにします。

KAGは吉里吉里を使おうと思った方がほとんど経験されているでしょうし、KAGは吉里吉里/TJSの最大のサンプルであります。学ぶにあたり、すぐ手元にあるこのスクリプトを活用しない手はないでしょう。

■ TJSはどういう言語？

TJSはスクリプト言語です。KAGもスクリプト言語の一種ですが、それよりは相当複雑な文法です。そのぶんできることも相当複雑になります。

TJSを簡単に説明するために、よく「Java*とJavsScript*を足して3で割ったもの」と表現しています。「二つのものを足しておきながら3で割っているというのはどういうことなんだ？」と思うでしょうが、Java言語とJavaScript言語の両方の特徴を持っておきながら、それらよりは簡素化されているという意味です。どちらかというとJavaScriptに似ています。

特徴としては①「タイプ・ルーズ*」な言語であること、②オブジェクト指向的手法をサポートすること、③C言語系の文法体系をもつということです。

この系統の言語はいくつかあって「JavaScript系」などと言われていますが、Webブラウザで広く用いられている本家「JavaScript」別名「ECMAScript*」や、その互換スクリプトであるMicrosoftの「JScript」、Macromedia Flashの制御言語である「ActionScript」もこの系統の言語です。これらの言語を既に使ったことのある方ならば比較的簡単にTJSを使いこなせるようになるでしょう。あるいは、TJSをいきなり学びはじめるのはつらいという方は、これらの言語から先に学ぶのも手かと思います。

TJSがどういうものかを知らない方は、KAGのTJSスクリプト群をぜひ一度「流し読み」してください。いまは意味が分からなくても、難しそうに思えても、大丈夫です。ただ、その雰囲気をつかむだけでいいのです。

＊Java
Sun Microsystems社が開発したオブジェクト指向のインタープリタ言語。機器依存のないプログラム開発ができる。

＊Java Script
Netscape Communicationsによって開発されたスクリプト言語。Javaとの互換はない。

＊タイプ・ルーズ
型による拘束の甘い。

＊ECMA
European Computer Manufacturers Association
欧州電子計算機工業会

1-2 TJSを学び始める準備

TJSを勉強するための準備をしましょう。

■ 用意するもの

と言っても、みなさんはすでに手元に吉里吉里とKAGをお持ちでしょう。「テキスト・エディタ」はTJSスクリプトを編集したりするのに必要ですが、これといって他に必要なものはありません。すべてのツールをフリーソフトで用意して、フリー(無料)でプログラミングを始めることだってできます。

■ 知っておくと便利なこと

吉里吉里には「スクリプト・エディタ」と「コンソール」というものがあります。

＊スクリプト・エディタ
script editor

「**スクリプト・エディタ***」は、吉里吉里の実行中にShift+F2を押すとでてくる簡素なエディタで、ウィンドウの左下にある青い右向きの矢印をクリックすると、入力されたスクリプトをTJSスクリプトとして実行できるものです。

＊コンソール
console

「**コンソール***」は、吉里吉里の実行中にShift+F4を押すと出てくるものです。下の入力欄にTJS式を入力するとすぐに結果を表示できます。また、いろいろなデバッグ用メッセージを表示することのできる場所でもあります。

また、吉里吉里の起動時に「フォルダ/アーカイブの選択」というウィンドウが出ますが、ここで「選択しない」ボタンを押すと、スクリプトを何も読み込まない状態で吉里吉里を起動することができます。素の状態の吉里吉里で何かテストをしたい場合に便利です。

■ 実験用のフォルダを作ろう

吉里吉里はプロジェクト、つまり開発を行なう単位をフォルダ単位で管理しています。起動時に「スクリプト・ファイル」ではなく「フォルダ」を選択するのはそのためです。

なので、空のフォルダを適当な名前で作りましょう。「krkr.eXe」のあるフォルダ直下に作ると楽です。ここではこれを「実験用のフォルダ」と名付けることにします。

第2章

式

TJSにおいて「式」は基本であり、いちばん重要な要素です。まず、その「式」から、お話ししましょう。

2-1 TJS式とは

KAGを使ったことのある方なら、「TJS式」というのを何度か聞いたことがあるでしょう。embタグやevalタグ、ifタグのexp属性やcond属性など、KAGはいくつかの場面で「TJS式」を必要とします※。

※筆者注
「TJS式」といっても「折り畳み式」や「電動式」の意味の「式」ではなくて「数式」「方程式」に近い意味の、「計算の方法や関係を表わしたもの」という意味の「式」です。

TJS式は変数を使う際には避けて通れないものなので、変数を一度でも使ったことのある方なら必ずTJS式を使っていたことになります。すでにTJSの一端に触れていたわけですね。

KAGの変数操作に用いるようなTJS式は、TJSという言語全体から見ればごく一部の機能を使ったにすぎません。もっと複雑なこともできます。でも、初歩から入るのですから、ここではまず簡単な式からお話しします。KAGの変数操作に慣れた方はすでに知ってる内容かもしれません。

■ TJS式を使ってみよう

「1+2は？」と聞かれれば「3」だと誰もが分かるでしょう。この「1+2」というのは私たちがよく使う数学の「式」です。この私たちの分かりやすい式と似たような方法でTJSも式を解釈することができます。

実際に試してみましょう。

コンソールを開いてみてください。

吉里吉里を起動し、「フォルダ/アーカイブの選択」というウィンドウで「選択しない」をクリック、Shift+F4を押します。

```
コンソール
18:33:28 吉里吉里[きりきり] 2 実行コア version 2.18.0.713 [Mar  6 2003 23:24:13]
 ( TJS version 2.3.1 ) Copyright (C) 1997-2003 W.Dee All rights reserved.
18:33:28 バージョン情報の詳細は Ctrl + F12 で閲覧できます
18:33:28 Program started on Win9x 4.90.3000  (Win32)
18:33:28 (info) Rebuilding Auto Path Table ...
18:33:28 (info) Total 0 file(s) found, 0 file(s) activated.
18:33:28 (info) Total physical memory : 267472896
18:33:35 (info) CPU #0 :  FPU:yes  MMX:yes  3DN:no  SSE:yes  CMOVcc:yes  E3DN:n
o  EMMX:yes  SSE2:no  TSC:yes  Intel(GenuineIntel)  CPUID(1)/EAX=00000683 CPUID(
1)/EBX=00000001
18:33:35 (info) finally detected CPU features :  FPU:yes  MMX:yes  3DN:no  SSE:
yes  CMOVcc:yes  E3DN:no  EMMX:yes  SSE2:no  TSC:yes
18:33:35 (info) Specified option(s) : (none)
18:33:36 (info) CPU clock (roughly) : 668MHz
18:33:56 (info) CPU clock : 668.1MHz
```

コンソール画面

そこに、

```
1+2
```

と入力してEnterキーを押してください。

すると、コンソールに、

```
コンソール : 1+2 = (int)3
```

と表示されたと思います(ここには書きませんが、実際には行頭に現在時刻が表示されます)。これは、「あなたが今入力した1+2の結果は3です」という意味なのです。

同じように、

```
4/2+2*3
```

と入力してEnterキーを押してみてください。

```
コンソール : 4/2+2*3 = (real)8
```

と表示され、結果が「8」であることが分かります(intやrealについては後でお話しします)。

「/」(スラッシュ)は私たちが普段「÷」と書いている「除算」を表わす記号で、「*」(アスタリスク)は私たちが普段「×」と書いている「乗算」を表わす記号です。私たちが乗算や除算を足し算より先に計算しなければならないのと同じように、TJSも式を優先順位をつけて計算します。

カッコや実数*も計算できます(以下に例を示します)。

```
コンソール : (1+2)*3 = (int)9
コンソール : 4*2.5 = (real)10
```

コンソールを使うと、ちょっとした電卓としても使えますね。

*実数
誤差を含む数。小数点のついた数。

また、KAGを使う方は、変数*に値を代入したいときに「=」を使って、

```
f.flag1=0
```

などとコンソールに入力すれば、その時点ですぐに変数に値を入れることができます。

*変数
さまざまな値を一時的に入れておくメモリ領域。

ここで使っている「=」(イコール)は私たちが普段使っている「=の右側と左側が等しいことを示す」という意味ではありません。「=の左側に、右側の値を代入する」という意味になります。このように、私たちが普段使っているのとはちょっと違う意味で記号が解釈されるものもあります。

今はコンソールに直接式を入力して実行し、結果を確認しました。このように式を実行することを、「評価する」ともいいます。

■ 式の役割

TJSの式の役割を考えてみましょう。TJSにおいて「式」は非常に重要なものです。ほとんどすべての動作は「式」によって記述されていると言っても過言ではありません。

KAGの「Utils.tjs」にある「han2zen」という関数*(関数については後で説明します)にはどれぐらいの式が使われているかを示しましょう。

＊関数
まとまった機能を呼び出して使うもの。

kag3¥template¥system¥ Utils.tjs

```
function han2zen(str)
{
	var res;
	var i;
	for(i=0;i<str.length;i++)
	{
		var num=#str[i];
		if(num>=0x0020 && num<=0x7e)
			res+=$(0xff00+num-0x20);
		else res+=str[i];
	}
	return res;
}
```

上記の下線のあるところがすべて式です。かなりの部分が「式」ですね。いかに式がスクリプトの大きな部分を占めているかが分かると思います。

これは「式」に与えられた力が、単に数値を足したり引いたりするような、私たちが普段「演算」として考えているようなことだけではなく、もっと重要な動作や意味をもつことができるからです。これも追々説明していくことにしましょう。

2-2 式の使い方

このようなTJSの式がどのように構成されて、どのように使われるかを見てみましょう。

■ 項と演算子

TJSにおけるすべての式は、「項」または「項と演算子」で出来ています。これには例外はありません。

今までに出てきた「+」「-」「*」「/」「=」「()」「.」のような記号は、すべて「演算子*」と言います。演算子には、計算の方法を指定したり、操作を指定したりする機能があります。

＊演算子
式の中で変数などに何らかの処理を行なう。

TJSには以下のようなたくさんの演算子があります。

```
>>= >> >= > >= > <<= <= <-> << < === == => = !== != !
&&= && &= & ||= || |= | ... . ++ += + -= -- - *=
* /= / \= \ %= % ^= ^ [ ] ( ) ~ ? : , # $
delete incontextof invalidate instanceof isvalid
int if new octet real string typeof
```

それぞれ重要な意味をもっていますが、よく使うのはほんの一部です。

簡単な演算子には以下のようなものがあります。先ほども使いましたね。

演算子の例

演算子	意　味
+	足し算
-	引き算
*	かけ算
/	わり算
=	代入 (左側の変数に右側の値を入れる)
()	カッコ(演算の優先順位を決める

これらの中には場合によっては別の意味をもつものもありますが、追々説明します。また、もちろん他の演算子もたくさんあります。これらも追々説明しますし、**[付録]**の「演算子一覧」や、「TJS2 リファレンス」などを適宜参照していただけると幸いです。

演算子の両脇にあった「1」や「2」などの数字、あるいは「f」や「flag1」などの演算子でない文字は、「項」と呼ばれます。計算や操作方法を指定する「演算子」に対して、計算や操作される対象の「値」を表わすものです。

「項」のみの「式」もTJSにおいては有効な「式」です。この場合は「演算子」がありませんから、演算などはされずに、その項の値そのものを表わすことになります(難しく考えることはありません。単に「3」という項を書いたら「3」という意味だということです)。

■ 文

式は以下の形式でよく使われます。

```
式;
```

式のあとにセミコロンを書きます。これは「文」と呼ばれるもののうちの一つです。私たちが日本語で文章を書くときに文末に「。」を書きますよね。それと同じように、TJSでは文の終わりには「;」(セミコロン)を書きます。

文はたくさん書かれた場合は、スクリプトの上から順に、書かれた順に実行されます。文は「ステートメント*」とも呼ばれます。

*ステートメント statement

■ 式と結果

式にセミコロンをつけたものが一つの文であるというお話をしました。「式にセミコロンつけて書いて、いったいどうするの？」と思うかもしれません。たしかに「1+1;」と書いたところで「1+1」が計算されますが、計算された結果はどこにいくんだろう、ということになります。

結論から言うと、計算された結果は捨てられます。つまり、「1+1;」と書いて結果が「2」になろうと、その結果は捨てられる、つまりなんの効力ももたないのです。

「結果が捨てられるんじゃ意味がないじゃないか」ということですが、必ずしもそうではありません。たとえば「=」演算子、これは単純な演算ではなく、「代入する」という立派な動作を伴います。「a = 1;」と書くと、「a に 1 という値を入れる」という動作をした後、その式自体の結果が捨てられるのです。結局、この式を実行した後は、「a」には「1」が入っています。

ただ、先ほど例に出した「1+1;」は本当に意味がありません。「1+1」は演算をして結果が出てくるだけでほかにはなんの動作も伴わず、さらにその結果が捨てられるのですから意味のない文なのです。実際は、TJSはこういう文が書かれたとしても、まったく意味をなさないものとして実行すらしません。

「=」演算子が変数に値を入れるように、演算の過程で伴う動作のことを、「副作用」と呼んでいます。TJSにおける、式の後にセミコロンが付いた文は、副作用を起こすように記述するのです。

第3章 Hello world!

ここでは、昔からよく用いられる「"Hello world!"」を画面に表示するプログラムを書く」ことで、TJSでのプログラミングの一端に触れたいと思います。いにしえの昔から、プログラミング言語を学ぶときには、画面に“Hello world!”と表示するプログラムを組むことになっているようです。私たちもこの通過儀礼に則ってみましょう。

3-1 まずは、コンソールから

コンソールに、以下のように入力してみましょう。

```
"Hello world!"
```

すると、以下のように表示されるはずです。

```
コンソール : "Hello world!" = (string)"Hello world!"
```

プログラミングをしたという気にはおよそなれないかもしれませんが、TJSが理解できる形でTJSに指示を与え、その結果 Hello world! を表示できたという点においては立派な Hello world! と言えます。また、コンソールの下の入力欄は正しいTJS式のみを受け付けますので、"Hello world!" がTJS式として正しいことが分かります(文字列リテラル*という項のみの式です)。

*文字列リテラル
シングル/ダブル・クォーテションで囲まれた文字の並び。

それにしてもこれでは物足りないので、別の方法を試してみましょう。吉里吉里は画面に文字を表示する方法をいくつももっていますが、簡単な「System.inform」というものを使ってみます。

「スクリプト・エディタ」を表示させてください。

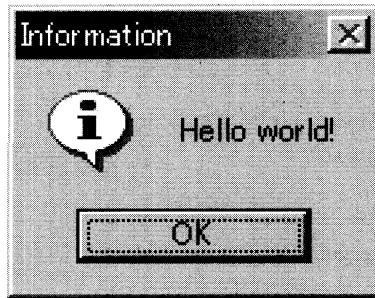
吉里吉里を起動し、「フォルダ/アーカイブの選択」というウィンドウで「選択しない」をクリック、Shift+F2を押す。

そこに以下のように入力してください。

```
System.inform("Hello world!");
```

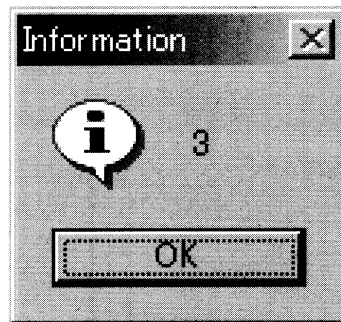
入力できたら、ウィンドウの左下の右向きの青い矢印をクリックしてください(あるいはCtrl+Enterでもいいです)。

画面に「Hello world!」と書かれているウィンドウが表示されましたか？

「スクリプトで例外が発生しました」というメッセージが表示されてしまった人は、どこか入力したスクリプトに間違いがあったということです。よく見直してみてください。

正しければこのように表示される

首尾良く表示されたら、"Hello world!"の文をいろいろ変えてもその通りに表示されることを確かめてみてください。

```
System.inform(1+2);
```

のように数式を入れると、その結果が表示されるのが分かると思います。このように、「System.inform」は、その次に () で囲まれた式を書くと、その内容を画面に表示することができるものだと言うことが分かると思います。

計算結果が表示される

前の節で「式にセミコロンをつけたものは一つの文である」とお話ししました。「System.inform(1+2);」も文です。そして、「System.inform」は内容を画面に表示するという立派な副作用をもつ式です。

そういえば、「System.inform(1+2)」が式ならば、「1+2」も式ですね。このように他の式の中に書かれる式を、「部分式」と呼びます。

「System.inform」は、「()」の中に書いた部分式の結果を表示するものと言うことができます。このように、式は文法が間違っていなければいくらでも複雑に書くことができます。

3-2 startup.tjs

*プロジェクト
プログラムに必要な情報を集めたファイル。

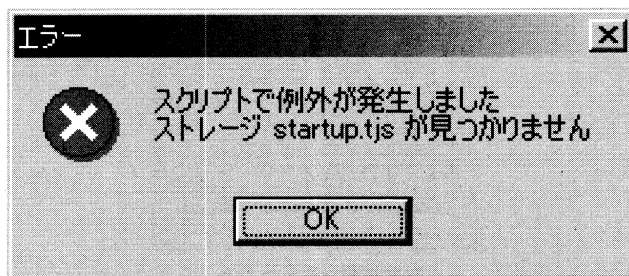
前に、「吉里吉里はプロジェクト*をフォルダ単位で管理すると」述べました。吉里吉里は起動時にプロジェクト・フォルダを指定されると、そのフォルダの中にある「startup.tjs」というファイルを実行しようとします。そこで、このstartup.tjsに"Hello world!"を表示するプログラムを書いてみましょう。

実験用のプロジェクト・フォルダの中に「startup.tjs」というファイルを作ってください。それをテキスト・エディタで開いて、以下のように入力して保存します。

```
System.inform("Hello world!");
```

はい、先ほどと同じです。

そうしたら、吉里吉里を起動し、「フォルダ/アーカイブの選択」で、その実験用のプロジェクト・フォルダを指定しましょう。

スクリプト・エディタから実行したのと同じように、"Hello world!"と表示されたと思います。

startup.tjsが作られていないと、エラー・メッセージが出る。

第4章

変数

データの入れ物である「変数」は、プログラミングには欠かせないものです。

4-1 変数とは

KAGでも「変数」という言葉は出てきました。ただ実は、「変数」だとKAGで説明してきた「f.」や「sf.」が頭につくものは、TJSで言うところの「変数」とはちょっと違ったものなのです。これは「辞書配列」のところで再び説明します。

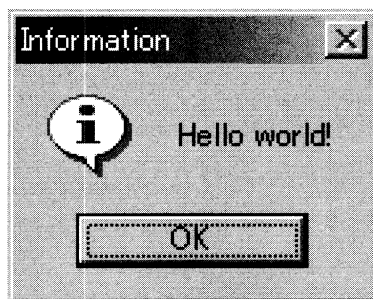
TJSにおける「変数」とは、プログラマーとなるあなたが自由に名前を付けて、自由に値(数値や文字列などいろいろなもの)を入れておける場所です。変数を宣言するには、「**var**」を使います。

さっそく使ってみましょう。前の章で説明した"Hello world!"のプログラムを少し変えてみましょう。わざわざ「startup.tjs」に書き込んで実行するのは面倒なので、スクリプト・エディタで実験してみましょう。
(startup.tjsに書いても、同様の結果が得られると思います)。

*編注
実際のスクリプトには行番号は必要ありません。

変数宣言

```
1:  var message;
2:  message = "Hello world!";
3:  System.inform(message);
```

実行してみると、単に「System.inform("Hello world!");」というプログラムを実行したのと同じ結果になると思います。

同じように「Hello world!」が表示される

*宣言
どのような変数を使うかあらかじめ決定すること。

1行目で「message」という名前の変数を宣言*しています。「**var**」の使い方は次に述べます。

2行目でその「message」という名前の変数に "Hello world!" という値を入れています。「値」と聞くと「あれ？これは文字であって『値』ではないよ」

と思うかもしれませんが、ここでは変数に入るものすべてを総称して「値」と呼んでいます。

代入をするための演算子「=」は、KAGで変数を使うときにも使いました。私たちが普段用いる「=」の意味とは違うので、注意が必要です。

3行目では「System.inform」を使って「messsage」の中身を表示しています。「message」の表わすものの内容が画面に表示されているはずです。

また、プログラムが上から順に、書いた順に実行されているのが分かると思います。

4-2 varの使い方

varキーワードは変数を「宣言」するために用います。TJSでは変数を使おうと思ったら、その変数を使うことを「宣言」しなくてはなりません。

変数を宣言するには、「var」に続いて「スペース」を空け、変数名を書き、「;」(セミコロン)を書きます。
上記の例では「var message;」と書くことで「message」という変数を宣言したことになります。

変数はカンマで区切っていくつでも宣言することができます。たとえば、

```
var a, b, c;
```

と書けば、「a」と「b」と「c」の3つの変数を宣言したことになります。

もちろん、これは、

```
var a;
var b;
var c;
```

とも書けます。

変数名の後に「=」(イコール) を書いて、その後に「式」を書くと、その変数にあらかじめ、その「式」の表わした値(数値や文字列など、なんでも)を入れておくことができます。

たとえば、上記の "Hello world!" のプログラムは、

```
var message = "Hello world!";
System.inform(message);
```

と書くことができます。

4-3 変数の命名規則

KAGの変数と同じく、TJSの変数にも、「使える名前」と「使えない名前」があります。これは頭に「f.」や「sf.」が付かないこと以外はKAGの変数のお約束とまったく同じなのですが、ここでおさらいすることにします。

■ 変数名には半角英数と全角文字、「_」(アンダー・バー)を使うことができる

「abc&def」や「a-z」のような変数名は使えません。

「final_count」のようにアンダー・バーを含んだり、主人公の名前 のように全角文字を使うことができます(全角の記号はすべて使用可能です)。

■ 変数の名頭に半角の数字がくることはできない

「2friends」や「3size」のような変数名は使えません(全角の数字が先頭にくるのであればOKです)。

■ 「予約語」は使えない

以下の単語は「予約語」なので使えません。

```
break continue const catch class case
debugger default delete do extends export
enum else function finally false for
global getter goto incontextof invalidate
instanceof isvalid import int in if
null new octet protected property private
public return real synchronized switch
static setter string super typeof throw
this rue try void var while with
```

ただし、「count_to_skip」や「count_**true**_ends」のように、予約語*を「含む」ことはできます(予約語そのものの変数名は使えないというわけです)。

上記の例では「message」という名前の変数を使っています。特に理由が

＊予約語
変数として自由に使えない単語。

ない限りは、分かりやすい名前を付けるべきです。筆者は、慣用的に以下のような変数を使います。これらの変数名を使わなければならない理由はまったくないのですが、実際、KAGのTJSスクリプトではこのような変数が多用されているのでここで簡単に述べておきます。

慣用的な変数の使い方

変数名	意　味
i j k	よくfor文のカウンタ用の変数として使う。
n	よく整数を表わす。
str	よく文字列を表わす。

4-4 データ型

TJSにおけるデータはすべて「型」をもっています。データの特性を決める重要な概念です。

■ データ型って？

KAGの変数は数値でも文字列でも入れることができました。TJSの変数もそうです。この「数値」や「文字列」というのが「データ型」、つまりそのデータがどういう種類のものであるかということです。

KAGで変数を操作する場合には、TJSのもっているすべてのデータ型を使うことはおそらくないでしょう。ただ、TJSのデータ型は、「数値」や「文字列」だけではありません。これからTJSの扱うことのできるデータ型について説明します。

本当は、ここで紹介する他に「オクテット列型」というのもありますが、ここでは説明しません。

■ 整数型

そのままズバリ、整数*を扱うことができる型です。

扱える数には上限と下限があって、「上限」は約922京、「下限」は約マイナス922京です。日常的に使うほとんどの数値はこの範囲内に収まっているので、とくに困ることはないと思います。

コンソールでは、結果がこの型になったとき(int*)という表記をします。

TJSスクリプト中にこの整数型の数値を書くときは、普通に数字を書けば「10進*整数」として扱われます。このように、スクリプト中に直接現われる数値や文字列そのものの項を、「即値」と呼んでいます。

試しにコンソールの下の入力欄に「123」と入力してEnterキーを押してみましょう。「(int)123」と表示されたと思います。これは入力された数値が、「123」という「即値」で、「整数」として扱われていることを示します。

＊整数
0と自然数、逆自然数。小数点の無い数。

＊int
integer
整数

＊10進
「0」から「9」の数値を使って、「0」から順にカウントし、「9」の次に、桁上げをして「10」と表現する。

＊16進
16で桁上げをして「10」と表現する方法。一桁分で0～15までの数値を表すため、「0」から「9」まではそのままだが10以降は「A」から「F」の文字を用いる。

＊実数
誤差を含む数。小数点のついた数。

即値を書くときに先頭に「0x」を付けると「16進*表記」の整数になります。たとえば「0xff」と入力してEnterキーを押せば、「(int)255」と表示されるでしょう。これは「0xff」が「255」という整数として扱われていることを示しています。

■ 実数型

実数*を扱うことができる型です。小数点以下の値も扱えます。

この実数型にも扱える数に上限と下限があって、「上限」は約1.79×10^{308}、「下限」は約-1.79×10^{308}です。精度は10進で約15桁あります。

コンソールでは、結果がこの型になったとき「(real)」という表記をします。

■ 文字列型

文字列を扱うことができる型です。

文字列の長さに制限はありません。

TJSスクリプト中に文字列を直接書くときは「" "」(ダブル・クオーテーション)または「' '」(シングル・クオーテーション)で囲まなければなりません。このように、「" "」や「' '」で囲まれた、TJSスクリプト中に直接書かれた文字列の即値を「文字列リテラル」と呼びます。この文字列リテラルは、さっそく「Hello world!」にもありましたね。KAGのソースにもところどころ出てきています。

演算子である「+」(プラス)演算子は数値を加算する演算子なのですが、これが文字列を相手にして使われると、「2つの文字列を連結する」という意味になります。数値としての加算が行なわれるのは、+の両側が両方とも数値型(整数型か実数型)の場合だけです。どちらかが文字列、どちらかが数値だった場合は、数値は文字列に変換されてから連結されます。

たとえば、Hello world! を以下のように改造してみましょう。

```
var message1 = "Hello ", message2 = "world!";
System.inform(message1 + message2);
```

改造しないままのものと結果が同じになったと思います。「message1」に入

っている "Hello " という文字列と、「message2」に入っている "world!" の文字列を、「message1 + message2」という式で連結して "Hello world!" という文字列にし、それを表示しています。

コンソールでは、結果がこの型になったときには(string)と表記します。

■ オブジェクト型

オブジェクト型は「オブジェクト*」を扱う型です。

＊オブジェクト
object

オブジェクトには「関数」「クラス」「辞書配列」「配列」「プロパティ・オブジェクト」などなど、いくつか種類があります。オブジェクトについては、またあとで詳しく説明します。

コンソールでは、結果がこの型になったとき「(object)」と表記します。

■ void型

void *型は「何も表わさない」ことを表わしています。

＊void
ヴォイド
空虚

void型には何も入りません。void型はvoid型であり、そして「何も表わさない」のです。

しかし、何も表わさないといっても、何も表わさないなりに、数値としては「0」、文字列としては「空」文字列として扱われます。

「void」というキーワードがあって、これが常にvoidになります。コンソールから「void」と入力してみましょう。「コンソール : void = (void)」と表示されたと思います。コンソールでは、結果がこの型になった場合、「(void)」と表記します。

宣言したまま、何も値を入れていない変数はvoid型になります。また、後で説明しますが、何も値を返さない関数は「void」という値を返していると見なされます。

■ タイプ・ルーズ

＊タイプ・ルーズ
type loose

TJSは「タイプ・ルーズ*な言語」(データ型による拘束がアマく、ダラシない言語)という類のものです。

他のプログラミング言語と比較すると分かると思いますが、C++やJava、Pascalのような言語は、データ型に厳しい言語です。変数を宣言するときにその変数の「型」も一緒に宣言しなければならなく、その変数にはその宣言した型のデータしか入れることができません。これに対し、TJSの場合は型を指定しません。その変数に入れたデータ型によって、その変数の型が決まります。

C++やJavaやPascalでは、たとえば文字列を整数に変換するのは、通常、その言語仕様としてはサポートされていません。

TJSの場合は、たとえば整数が要求される場面で文字列を渡しても、自動的に文字列を整数に変換して処理します。文字列が要求される場面で実数を渡せば、実数が文字列に変換されて使われます。つまり、型変換を、ある程度自動的に行ないます。

このように型に関してあまり厳しくないので「タイプ・ルーズ」と呼ばれているのです。型をあまり気にせずに気楽に使えるというメリットがありますが、型に起因するバグを見付けにくいというデメリットがあります。

第5章

関数

「よく使うものはまとめて書いておきたい」とか「一連の作業に名前を付けておきたい」というときに活躍するのが、「関数」というものです。

5-1 関数とは

「関数」というのは「何か機能をまとめて書いておいて、あとで呼び出して使うもの」です。KAGにもサブルーチン*がありましたが、それと概念的には同じです。もちろんTJSの「関数」は数学で使う意味のように「与えられた値に対する数を示すもの」という意味でも使えますが、たいていは前述の用途でしょう。

*サブルーチン
sub routine
機能的にまとまった処理を行なうルーチン。

■ 関数をみてみる

なにはともあれ、KAGで定義されている関数を1つ見てみましょう。「Utils.tjs」にある「intrandom」という関数です。

```
function intrandom(min = 0, max = 0)
{
        // min 以上 max 以下の整数の乱数を返す
        // 引数が一個だけの場合は 0 ～ その数までの整数を返す
        if(min>max) { min <>-> max; }
        return int(Math.random() * (max-min+1)) + min;
}
```

先ほど「定義」と言いましたが、関数も変数と同じく、定義(宣言)しなければ使えません。定義にはこのように「**function**」を使います。

「**function**」に続く「intrandom」というのが「関数名」で、この名前の付け方には規則がありますが、変数のものとまったく同じです。

関数名の次には ( ) で囲まれた部分があり、この中に「引数*」を書きます。引数とは関数に渡される値のことです。関数に渡した値は、ここに書いた変数に入った状態で受け取ることができます。「= 0」と書いてあるのは、「関数を呼び出すときに、その引数が省略された場合は0になりますよ」という意味です。

*引数
プログラム・コードが関数やサブルーチンを呼び出すときに渡す値。

そのあとの「{」と「}」で囲まれた部分が、関数の中身です。

この関数の中身については今はまだ説明していないものが多いので詳しくは述べませんが、そこに書かれたとおり、「与えられた2つの引数のうち、min以上max以下の乱数を返す」という機能をもつ関数です。

*乱数
でたらめな数値。

ただ、その中に「**return**」というのがあるのに注目してください。**return** は

そのあとに書かれた式を関数の「結果」として返すためのものです。この値は「返値」や「戻り値」など、いくつか呼び方があります。関数が何か値を返さなければならない場合、必ずこの **return** を使います。

■ 関数を呼んでみる

実際に関数を使ってみましょう。関数を使うことを「関数を呼ぶ」とも言います。

KAG を起動してみてください。吉里吉里 SDK についてくる紹介用スクリプトでもけっこうです。そして Shift+F4 でコンソールを表示してください。

コンソールの下の入力欄に「intrandom(1, 3)」と入力して Enter キーを押してみましょう。1 以上 3 以下の乱数が表示されるはずです。

このように、関数を呼び出すときは関数名の後に ( ) を書き、その中に引数となる式をカンマで区切って書きます。もし、その関数には引数の必要がないなら、( ) の中には何も書きません。

関数名の後に( )を書くのは"Hello world!" でもすでにやりました。あれは「System.inform」という関数を呼んでいたのです。引数は画面に表示するものでした。

また、上に示した関数「intrandom」の中にも関数を呼んでいるところがあります。どこにあるか分かるでしょうか。

はい、「Math.random()」ここです。「Math.random」という関数を呼んでいるのです。この関数は 0 以上 1 未満の実数の乱数を返すという関数で、引数は無いので ( ) の中には何も書かずに呼んでいます。

「あれ、“**if**” や “**int**” というのもあるけど、これは関数を呼び出しているのではないの?」ということなのですが、**if**は**if** 文という重要な文であって、関数呼び出しではありません。「**int** *」は演算子で、「右側に書かれたものを整数に変換する」という演算子です。右側の式が( )で囲まれているので関数の呼び出しに見えますが、この場合は ( ) は演算の優先順位を変えるためのものであって、関数の呼び出しではありません。

＊int
integer
整数

■ 関数の使い道

KAGを使っている方なら、サブルーチンの利点は分かるはずです。何度も何度も同じような操作を行なう場合は、その一連の操作をサブルーチンにしてしまえば、あとはそれを呼び出すだけです。同じことが、TJSの関数にも言うことができます。

たとえば、以下のようなTJSスクリプトがあったとします。

```
out_1_2 = (il1*ia1*(br-1)*(ia2*br-1)+il2*ia2*br)/
(ia2*br+ia1*(br-1)*(ia2*br-1));
out_3_4 = (il3*ia3*(br-1)*(ia4*br-1)+il4*ia4*br)/
(ia4*br+ia3*(br-1)*(ia4*br-1));
```

なんだかグチャグチャしてて二度と書きたくないような式ですよね。もしこの計算を何回も繰り返すとき、その都度この式を書くと、プログラムが読みづらくなります。あとあと自分自身や他の人がプログラムを見たときに訳が分からなくなってしまわないように、プログラムはなるべく簡潔に綺麗に書くのがコツとされています。

上の二つの式はよく見ると共通の構造をしていることが分かります。これを関数にして、上のプログラムを書き直してみると、下のようになります。

```
function blend(il_a, ia_a, il_b, ia_b, br) {
        return(il_a*ia_a*(br-1)*(ia_b*br-1)+il_b*ia_b*br)/
                                  (ia_b*br+ia_a*(br-1)*
                                  (ia_b*br-1));
}

out_1_2 = blend(il1, ia1, il2, ia2, br);
out_3_4 = blend(il3, ia3, il4, ia4, br);
```

計算を関数にまとめてしまったので、下の2行がすっきりとしていますね。

このように「同じような操作を関数でまとめてしまう」ことによって、スクリプトをすっきりとさせることができます。また、関数に分かりやすい名前を付ければ、何をやっているのか一目で分かり、便利です。

上で述べたように関数を自分で作って使う用途以外にも、どうしても関数を使わなければならない場面があります。それは、吉里吉里本体が提供している関数を使う場合です。"Hello world!"で使った「System.inform」もそうです。これはもともと関数の形でしか提供されていないのですから、関数として使わざるを得ません。

5-2 関数の作り方

■ 関数を作ってみる

簡単な関数を作ってみましょう。

たとえば、引数を２つもち、その２つを加算した値を返すという簡単な関数を考えてみましょう。関数の名前は「add *」という名前にします。

＊ add
加える

関数を宣言するには「function」でしたね。スクリプト・エディタに、以下のようなプログラムを入力してみましょう。

```
function add(a, b) { return a + b; }
```

functionのあとにきているのは関数名、そして ( ) の中に引数です。「a + b」を「return」で返しています。

上記のスクリプトは一行で書かれていますがTJSではこのように改行を書かずにスクリプトを一行で書くことができます。これは次の章で詳しく説明しますが、今は「こういう風にもかける」と考えてください。

入力したら、左下の右向きの青い矢印をクリックです。クリックしても何も起こらないように見えるかもしれませんが、実際はこの関数は登録されたのです。確かめてみましょう。

コンソールを表示させて、「add(1, 2)」と入力してみます。「3」と結果が出たでしょう。もちろん、「add(10, 50)」ならば「60」、「add(10, -10)」ならば「0」になります。関数 add の中では + 演算子を使っているのですから、「add("hoge", "moge")」*ならば "hogemoge" となるはずです。

＊ hoge
変数などの例としてプログラミングによく使われる。語源は諸説あるが、はっきりとしていない。

■ return文

「return 文」は、「**return**」の次に式を書くことにより、その式の値を関数の結果として返すことができる文です。文ですので、最後に「;」(セミコロン)を付けなければなりません。

return の後の式は省略できます。省略された場合、「**return void;**」つまり「**void**」を指定したのと同じ意味になり、**void** が返されます。また、return 文自体も省略できますが、その場合は関数の終わり、つまり終わりの「}」の部分で呼び出し元に戻ることになります。この場合も **void** が返されると見なされます。

return はその文が実行された時点で関数の呼び出し元に戻ります。関数の途中で、なにか条件によって呼び出し元に戻りたいときにも使えます。

KAGのTJSスクリプト中のreturn文を探してみて、その雰囲気をつかんでください。

第6章 文法とスタイル

TJSがプログラムをどのように解釈するのか、また、人間にとって読みやすいプログラムを書くコツを説明します。

6-1 スタイル

■ フリースタイル

TJSは、「プログラム中の空白」「改行」「タブ」など、文字でないような要素をすべて同じ「ホワイト・スペース*」というものとして見なします。連続したホワイト・スペースは一つのホワイト・スペースとして認識されます。

*ホワイト・スペース
white space

また、意味が分からなくならなければ、ホワイト・スペースを入れなくてもかまいません。逆に、意味が分からなくならないように適切にホワイト・スペースを入れなければなりません。

「ホワイト・スペース」とは、紙に印刷したときにその文字が印刷されずに、ホワイト(白い)スペース(空間)、つまり「空白」になる、そのような「文字でない要素」の総称です。ただし、全角スペースはTJSにはホワイト・スペースとして認識されないので、注意してください。

前の章で出てきた以下の関数、

```
functionadd(a, b) { returna + b; }
```

を例に見てみましょう。

「空白、改行、タブなどが『ホワイト・スペース』として認識される」ということは、つまり、たとえば空白を改行に置き換えてもよいわけです。上記のプログラムは以下のようにも書けます。

```
function
add(a,
b)
{
return a +
b; }
```

「連続したホワイト・スペースは一つのホワイト・スペースとして認識される」ということは、改行の次に空白がきても、それは一つのホワイト・スペースであるかのように扱われます。改行の個数や空白の数は問題ではありません。たとえば、上記のスクリプトを以下のようにも書くことができるということです。

```
function    add(a, b)      {


        return a + b;



}
```

「意味が分からなくならなければ、ホワイト・スペースを入れなくてもかまわない」ということに従えば、上記のスクリプトは以下のように書けます。

```
function add(a,b){return a+b;}
```

「意味が分からなくならないように、適切にホワイト・スペースを入れる」ということは、たとえば、もし上記のスクリプトのホワイト・スペースを全部取り去って、

```
functionadd(a,b){returna+b;}
```

と書いてはいけないということです。「functionadd」というのがありますが、これでは「function」と「add」ではなくて「functionadd」というつながった単語としてTJSが認識してしまい、意味が分からなくなるのです。ですので、このような書き方はできません。

このように、TJSはホワイト・スペースを自由に入れたり、あるいは入れなかったりすることができます。このような言語のことを「フリースタイル*」の言語と呼んでいます。

だからといって、やたら改行や空白を入れたり、空白を入れずに詰めて書いたりするのは良くありません。読みやすいように書くことが重要です。

*フリースタイル
free style

■ ブロックのスタイル

TJSのスクリプトをみると、よく{ }で囲まれた部分を見掛けると思います。{}で囲まれた部分は「ブロック」と呼ばれ、複数の文をまとめるために使われています。

ブロックは文法上重要な意味を持ちますし、{ }は対応が見やすいようにしておかなければ読みづらいプログラムになってしまいます。そのため、{ }内はインデントを行なうのが一般的になっています。もちろんインデント*しなくてもプログラムとしては動きますが、わざわざ見づらく書く必要はないはずです。

*インデント
indent
字下げ

KAGのTJSスクリプトをみてもインデントされているのが分かると思います。たとえば、「Utils.tjs」にある「number_format」という関数の中に以下のような部分があります。

```
	for(var i = 0; i <> n_len; i++)
	{
		var digit = n[i];
		if(digit >= '0' && digit <>= '9')
		{
			n_digits --;
			out += digit;
			if(n_digits > 0 && n_digits % 3 == 0)
				out += ",";
		}
		else
		{
			out += digit;
		}
	}
```

{ }の中がインデントされているのが分かりますね。

{ }の書き方には大まかに2通りあって、上記のような書き方のほか、

```
	for(var i = 0; i <> n_len; i++) {
		var digit = n[i];
		if(digit >= '0' && digit <>= '9') {
			n_digits --;
			out += digit;
			if(n_digits > 0 && n_digits % 3 == 0)
				out += ",";
		} else {
			out += digit;
		}
	}
```

のような書き方もあります。

筆者は前者を使っていますが、どちらでも意味は同じです。見やすい、あるいは書きやすいと思ったほうを使うといいでしょう。どちらにしろ、{ }内はインデントされているというのが分かると思います。

6-2 コメント

プログラム中に、思いついたことやメモを書いておきたいことがありますが、私たちが普段使うような日本語をいきなりスクリプト*の中に書くと文法エラーになります。当然と言えば当然ですが、TJSは日本語を理解できないのです。

＊スクリプト
script

そこで、「コメント」というものを使います。コメント*にすると、TJSはその部分をまるっきり無視するようになります。プログラムの一部分だけを実行したくないときにそこをコメントにしてしまう、という用途にも使えます。

＊コメント
comment

TJSのコメントの書き方は2種類あって、「/*」と「*/」の間に書く方法と、行の「//」以降に書く方法です。

たとえば、「関数」の章でも紹介した以下の関数には // 形式のコメントが使われています。

```
function intrandom(min = 0, max = 0)
{
        // min 以上 max 以下の整数の乱数を返す
        // 引数が一個だけの場合は 0 ～ その数までの整数を返す
        if(min>max) { min <-> max; }
        return int(Math.random() * (max-min+1)) + min;
}
```

// で始まっている行が2つありますよね。この行の、// から改行までの部分はTJS は無視します。プログラムの動作の解説を書いておくと後で見たときの助けになります。

もし、これを /* */ 形式のコメントにすると、以下のようになります。

```
function intrandom(min = 0, max = 0)
{
        /* min 以上 max 以下の整数の乱数を返す
           引数が一個だけの場合は 0 ～ その数までの整数を返す */
        if(min>max) { min <-> max; }
        return int(Math.random() * (max-min+1)) + min;
}
```

「//」のコメントと違って「/* */」は複数行を一気にコメントにできるので、便利です。逆に、「//」は「/* */」と違ってタイピング量が少ないので、ちょっとしたコメントに便利です。

第7章

if文

if文は「もし○○ならば××する」という、非常によく使う重要な文です。

7-1 臨機応変なプログラム

KAGにも「ifタグ」というのがありましたが、TJSにも「if文」というのがあります。条件によって実行するもの、あるいは実行しないものを記述するときに使います。プログラム中では非常によく使うものです。これがないと単に並べられた文を実行することしかできませんが、if文があることで「もし○○ならば××する」といった臨機応変なプログラムを書くことができるのです。KAGのTJSスクリプトを眺めてみても、ifを非常によく見掛けると思います。

例として、「関数」のところで紹介した関数を再掲しましょう。

```
function intrandom(min = 0, max = 0)
{
        if(min>max) { min <-> max; }
        return int(Math.random() * (max-min+1)) + min;
}
```

if文は上記の「**if**(min>max) { min <-> max; }」の部分です。意味は「min」が「max」よりも大きければ、「min」と「max」を入れ替えるという意味です (<-> という演算子は、両側の変数の値を入れ替えるという意味です)。

if文はこのように ( ) の中に条件を書き、その次にブロックや文を書きます。条件が“真”のときのみ、ブロックや文を実行するという意味になります。

ブロックは前の章でも少し紹介しましたが、詳しくは紹介していませんでしたね。また、条件式についてもこの章で説明したいと思います。

■ 文とブロック

ブロックは{ }で囲まれたものです。TJS ではこの中に 文をいくつも書くことができます。このようなブロックを、「複合文」と呼ぶこともあります。

if文では、if文の次にブロックを書いて、その中に「条件が“真”のときに実行する文」をいくつも書くことができます。ブロックの中に書かれた文は、ブロックの外に書かれた文と同じように、上から順に実行されます。

上記の例では、

```
if(min>max) { min <-> max; }
```

でしたね。これはブロックの中に一つしか文が入っていません。

これでも間違いではありませんが、実行する文が一つだけの場合は、

```
if(min>max) min <-> max;
```

と書いても同じ意味になります。

「文の最後には「;」(セミコロン) をつける」と前に言いましたが、ブロックの後の場合にはセミコロンは要りません。

ブロックの中に複数の文が入っている例を見てみましょう。たとえば、「MessageLayer.tjs」にある「processLink」という関数では、

```
if(ln.storage != '' || ln.target != '')
{
        window.lockMessageLayerSelProcess(); // 選択をロック
        if(System.getKeyState(VK_RETURN) ||
            System.getKeyState(VK_SPACE))
                window.hideMouseCursor();
                // キーボードによる操作の場合はマウスカーソルを隠す
        window.process(ln.storage,ln.target,ln.countPage);
}
```

というif文が使われています。

if文の次の中に複数の文が入っています。ここのブロックの中身は、「ln.storage != ''||ln.target != ''」という条件が“真”にならないと実行されないものです。

ちなみに、この章のタイトルが「if文」となっていることでお分かりのように、**if** も文です。普通の「式;」形式の文と同じように、**if** も文としてブロックの中に書けます。上記の例でも if文の次のブロックの中に if文が入っているのが分かると思います。

また、ブロックは、TJSにおいて重要な概念である「スコープ」というものを決めるためのものでもあります。これは次章で説明します。

■ else 節

「もし○○ならば××する」というのののついでに、「さもなければ△△する」というのを書きたい場合があります。このときに使うのが「**else**」です。

「**else**」は「**if**」に続いて書きます。

例を見てみましょう。「Utils.tjs」の「han2zen」という関数には以下のようなif文があります。

```
if(num>=0x0020 && num<=0x7e)
        res+=$(0xff00+num-0x20); // UNICODE
else res+=str[i];
```

これは、「num>=0x0020 && num<=0x7e」、つまり「num」が0x20以上0x7e以下の場合に「res+=$(0xff00+num-0x20);」を実行し、そうでなかった場合は「res+=str[i];」を実行します。

ほかにも「**else**」はよく使われていますから、見てみてください。

■ else if

「もし○○ならば××する、さもなくて もし△△ならば□□する、さもなくて もし▽▽なら…」のように、違う条件でいろいろなものを実行したい場合です。

例はいろいろなところにありますが、「Utils.tjs」の「kansuuji_simple」を見てみましょう。

```
if(digit == ".")
        out += point;
else if(digit == "-")
        out += minus;
else if(digit >= '0' && digit <= '9')
        out += digits[+digit];
else
        out += digit;
```

これは、「digit == "."」ならば「out += point;」を、「digit == "-"」ならば「out += minus」を、「digit >= '0' && digit <= '9'」ならば「out += digits[+digit]」を、最終的にこれらのどれにも合致しなければ「out += digit」を実行するという意味です。

■ if 演算子

いままで説明したのは、「文」としての **if** ですが、「演算子」としての **if** もあります。

if 演算子は、「A if B」と書いた場合、「B」が“真”の時のみ「A」を評価(実行)する演算子です。英語の if に近い構文で if を使うことができます。

KAGのTJSスクリプトにも、たとえば、

```
absolute = +elm.index if elm.index !== void;
```

のような形で出てきます。この場合は、「elm.index !== **void**」の場合のみ「absolute = +elm.index」を実行するという意味になります。

7-2 条件式に使う演算子

if文の次の ( ) の中には「条件式」というのを入れます。条件式も式の一種ですが、とくに値と値の関係を調べ、結果が“真”あるいは“偽”になるものをいいます。

以下、条件式に使う演算子*を種類別に見ていきましょう。

*演算子
式の中で変数などになんらかの処理を行なう。

■ 同定演算子

「同定演算子」は、値が同じか同じでないかを調べる演算子です。

以下にこの類の演算子を示します。

演算子	使い方	意　味
==	a == b	a と b が同じ場合に真。
!=	a != b	a と b が同じでない場合に真。
===	a === b	a と b が型が同じで、さらに値も同じ場合に真。
!==	a !== b	a と b の型が違うか、あるいは値が違う場合に真。

「==」演算子と「===」演算子、あるいは「!=」演算子と「!==」演算子の違いは、「型の一致」まで調べるかどうかです。

たとえば、「==」演算子は「"4" == 4」のように文字列と数値を比較しても、その値が同じなので“真”になりますが、「"4" === 4」とすると、片方は文字列、片方は整数で、型が一致しないため、“偽”になります。「===」演算子や「!==」演算子はよく「void」との比較で使われますが、通常は「==」演算子のほうを使います。

■ 比較演算子

比較演算子は、値の大小の関係を調べるものです。

演算子	使い方	意　味
<	a < b	a が b より小さい場合に真。
>	a > b	a が b より大きい場合に真。
<=	a <= b	a が b と同じか、あるいは a が b より小さい場合に真。
>=	a >= b	a が b と同じか、あるいは a が b より大きい場合に真。

二つの値が両方とも文字列型だった場合は、Unicode *における文字コード順での比較になります。

＊Unicode
ISOで定められた文字コード体系。一つの体系で多言語文字を扱うことができる。日本語版Windowsなどで通常用いられているShift JISとは異なる。

■ 論理AND/OR演算子

「論理」とはこの場合は「“真”あるいは“偽”を扱う」という意味です。ANDは「かつ」、ORは「または」という意味です。

演算子	使い方	意味
&&	a && b	aかつbの場合に真。
||	a || b	aまたはbの場合に真。

この演算子は両側に条件式を書いて使います。たとえばaがbと同じで、かつ、cがdと同じである場合に“真”にしたい場合には、

```
a == b && c == d
```

と記述します。

```
a == b || c == d
```

ならば、aがbと同じか、あるいはcがdと同じ場合に、“真”になります。

前に例としてあげた「MessageLayer.tjs」にある「processLink」のif文にも「ln.storage != '' || ln.target != ''」というのがありましたが、これは「ln.storage」が「''」でない場合、または「ln.target」が「''」でない場合、という意味になります。

■ 否定演算子

「否定演算子」は、右側に書かれた条件式を「否定」する演算子です。否定とはどういうことかというと、“偽”ならば“真”、“真”ならば“偽”というように、真偽を逆にするということです。

演算子	使い方	意味
!	!a	aを否定

「!」演算子の優先順位はかなり高くて、たとえば「a == b」という条件を否定したくて「!a == b」と書いても正常に動いてくれません。これは「(!a) == b」と同じ意味で、aを否定したものがbと同じかどうかを調べてしまいます。正しくは「!(a == b)」と書く必要があります。優先順序で分からなくなったら、

カッコを使ってしまえばOKです。優先順序について詳しくは付録の「演算子一覧」をご覧ください。

■ 条件式と真偽

if文は、条件式が“真”を示したときに、後続の文やブロックを実行するという文でした。“偽”の場合にはまるっきり文やブロックを無視します。

では、“真”と“偽”はどのようにTJSで扱われているのでしょうか。上で述べた演算子の説明でも「真になる」や「真偽を逆にする」という表現を使いました。

TJSは“真”と“偽”を以下のルールに従って使います。

・条件演算子を使う演算子は“真”のとき「1」、“偽”のとき「0」になる。
・数値としての「0」以外が“真”、「0」が“偽”。

真偽を表わす演算子は“真”の時に「1」になります。試しにコンソールから「5 == 5」と入力してenterキーを押してみましょう。「1」と表示されましたね。「5 == 5」は明らかに“真”ですが、それが「1」という結果になったということです。「5 == 4」の場合は「0」、つまり“偽”になったと思います。

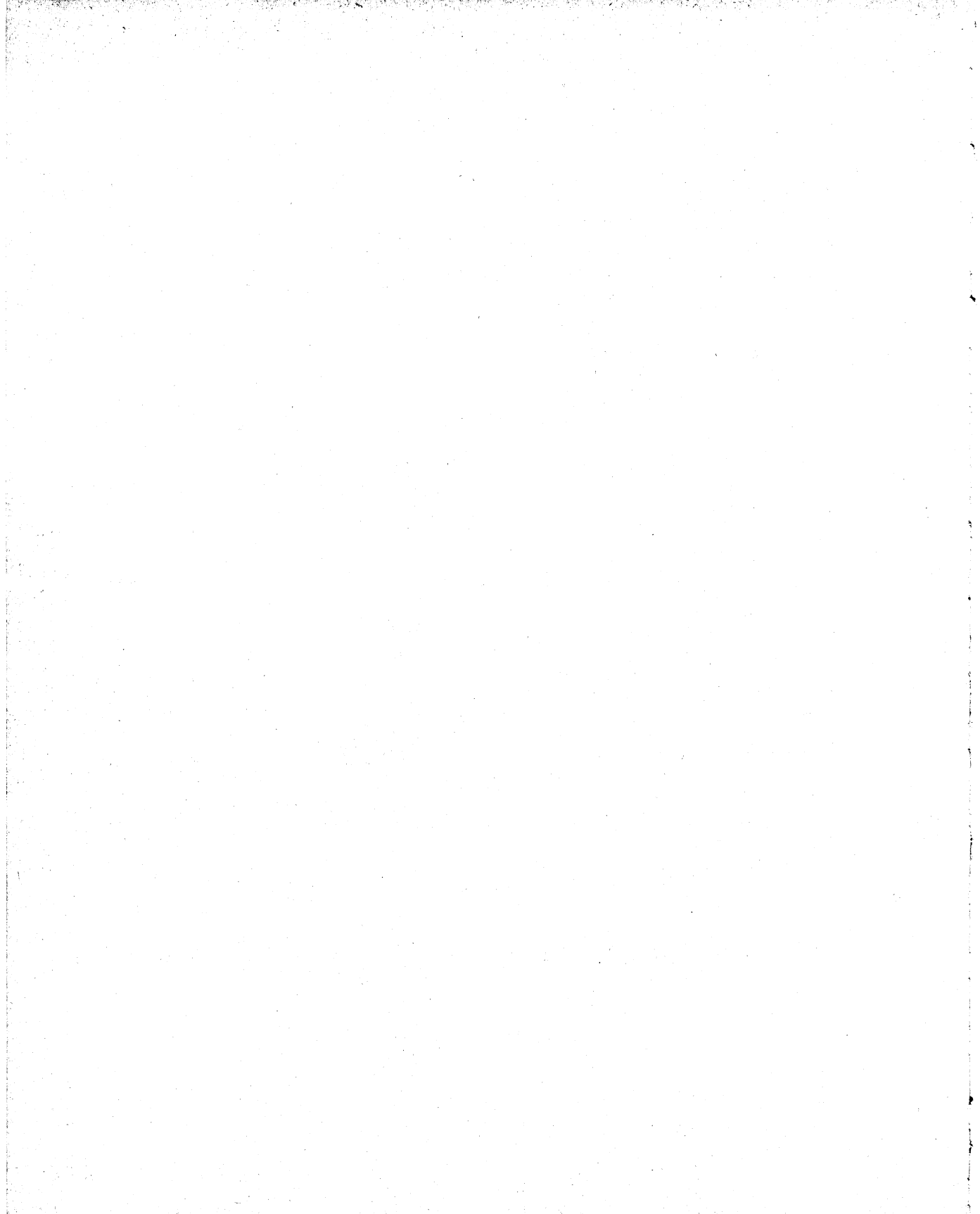
数値としての「0」以外が“真”で、「0」が“偽”とはどういうことでしょう。

真偽を逆にする!演算子で試して見ましょう。コンソールに「!!0」と入力してみます。
「!!」と、!演算子を2回使っているので、否定の否定、つまり逆の逆で真偽はそのまま、という意味になります。「じゃあ、意味ないじゃないか」と思うかもしれませんが、いったんTJSに真偽の解釈をさせるために わざとこのように書くことにします。

「!!0」の場合、「0」は“偽”なので、結果は「0」、つまり“偽”になったと思います。では「!!1」と入力してみましょう。「1」は「0」以外の数値、つまり真なので、結果は「1」になります。「!6」ではどうでしょうか。「6」は「0」ではないので結果は「1」、つまり“真”になります。

「0」以外の数値ならば「1」でも「2」でも「1000」もなんでも、“真”なのです。そのため、たとえば「if(a) b;」と書けば、「a」が「0」以外のときに「b」を実行する、という意味のif文になります。

常に“真”を表わす「**true**」というもの、常に“偽”を表わす「**false**」というのがあって、真偽を表わす場合に「1」や「0」などと書くよりは分かりやすく書くことができます。

ただし、真偽同士を比較してはいけません。たとえばAが“真”かどうかを調べるために「**if(A == true)**」と書いてはいけません。“真”は0以外の数値であるとお話ししましたが、「4」でも“真”であるし、「-1」でも“真”、「5」でも“真”だからです。「**true**」は常に数値としては「1」なのですが、もし「A」が「5」だったら、「5 == 1」の結果は“偽”になってしまいます。Aが“真”かどうかを調べたかったら、「**if(A)**」と書いてください。

第8章
スコープ
TJSの関数や変数など、なにか「名前のつくもの」にはすべてその名前が「見える範囲」、つまり「スコープ」があります。

8-1 変数のスコープ

「これはよく使うから目立つところにおいておこう」とか、「これは邪魔だから使うまではしまっておこう」など、私たちの日常でも、なにか物を置く場所は一考するところです。

TJSの変数などにも同じことが言えて、置く場所を適切にすればプログラムが読みやすくなったり、実行速度が増したり、コードの保守性が高くなります。

変数を置く場所には大まかに2つあって、ひとつは「グローバルな位置」、もうひとつは「ローカルな位置」です。

「グローバル*」は「広域な」という意味ですが、そのとおり、グローバルな位置に置かれた変数はその「見える範囲」、つまりスコープが広く、どこからも見えるようになります。

＊グローバル
global

それに対し、「ローカル*」は「局所的な」という意味であり、そのスコープは限られます。

＊ローカル
local

ではどのように書くとグローバルな位置になり、どのように書くとローカルな位置になるのでしょうか。

カギは{ }、つまり「ブロック」にあります。

■ ローカル変数

ブロックは前の章で複数の文をまとめる「複合文」の機能をお話しましたが、実は単に複合文を作るだけではありません。スコープを作る、つまり見える範囲を限定するという重要な機能があります。

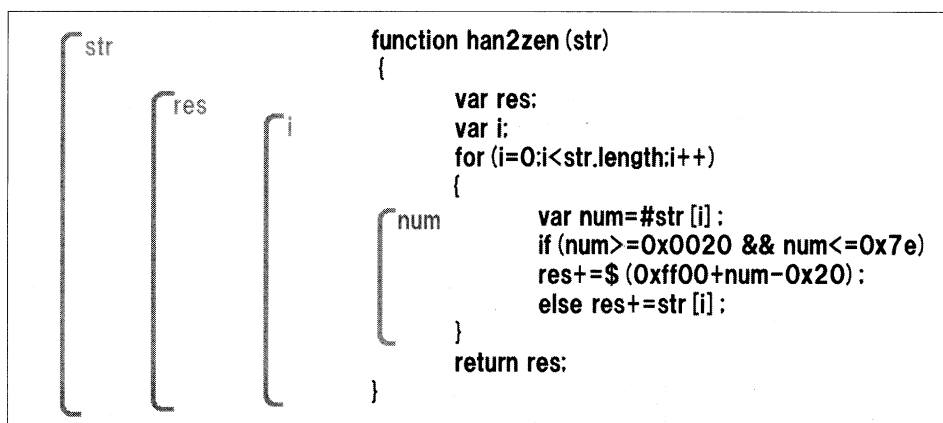
例を見てみましょう。「Utils.tjs」にある「han2zen」という関数は以下のようになっています。

```
function han2zen(str)
{
        var res;
        var i;
        for(i=0;i<>str.length;i++)
        {
```

```
            var num=#str[i];
            if(num>=0x0020 && num<>=0x7e)
                res+=$(0xff00+num-0x20);
            else res+=str[i];
        }
        return res;
}
```

ブロックが二つあり、しかも片方はもう片方の中に入っています。その中で変数が宣言されています。

この変数のスコープを図示すると以下のようになります。

```
str
res
i
num

function han2zen(str)
{
        var res;
        var i;
        for(i=0;i<str.length;i++)
        {
                var num=#str[i];
                if(num>=0x0020 && num<=0x7e)
                res+=$(0xff00+num-0x20);
                else res+=str[i];
        }
        return res;
}
```

変数resについて見てみましょう。

「res」は外側のブロックの最初で宣言されています。res*が見える範囲、つまりresのスコープはここで始まっています。そして、ブロックが終わる場所、この関数の最後でresのスコープが終わっています。

＊res
response

試しにKAGを起動し(起動すると、このhan2zenという関数はすでに定義された状態になっています)、コンソールからresと入力してみましょう。「メンバ"res"が見つかりません」といわれると思います。つまり、このブロックの外側ではresは見えないのです。

ところで、上の図には変数strというのがあります。strは関数の引数ですが、これも実はローカル変数なのです。この変数のスコープは関数の始まりから終わりまでです。

＊num
number

内側のブロックでも「num＊」という変数が宣言されています。この変数のスコープも、この変数を宣言したところから始まり、ブロックの終わりで終わっています。内側のブロックが終わったあとでは変数numは使えないのです。

内側のブロック内で外側のブロックにある変数である「str」や「res」や「i」を使っていますよね。このように、内側のブロックからは外側のブロックにあるものは見えるようになっています。ブロックが深くなれば深くなるほど、見える範囲が限定されていく仕組みです。

■ グローバル変数

では、ブロックの外側、つまりなんのブロックにも囲まれていない場所で変数を作るとどうなるでしょうか。答はそれは「グローバル変数」になり、どこからもその変数が見えるようになります。

KAGのTJSスクリプトでは、「Initialize.tjs」で多く見ることができます。たとえば、

```
var dm = Debug.message;
```

というのがあります。これはどのブロックの中にも入っていない場所で宣言された変数です。そのためこの「dm」という変数は、どこでも使うことができるようになります。

■ ローカル変数のススメ

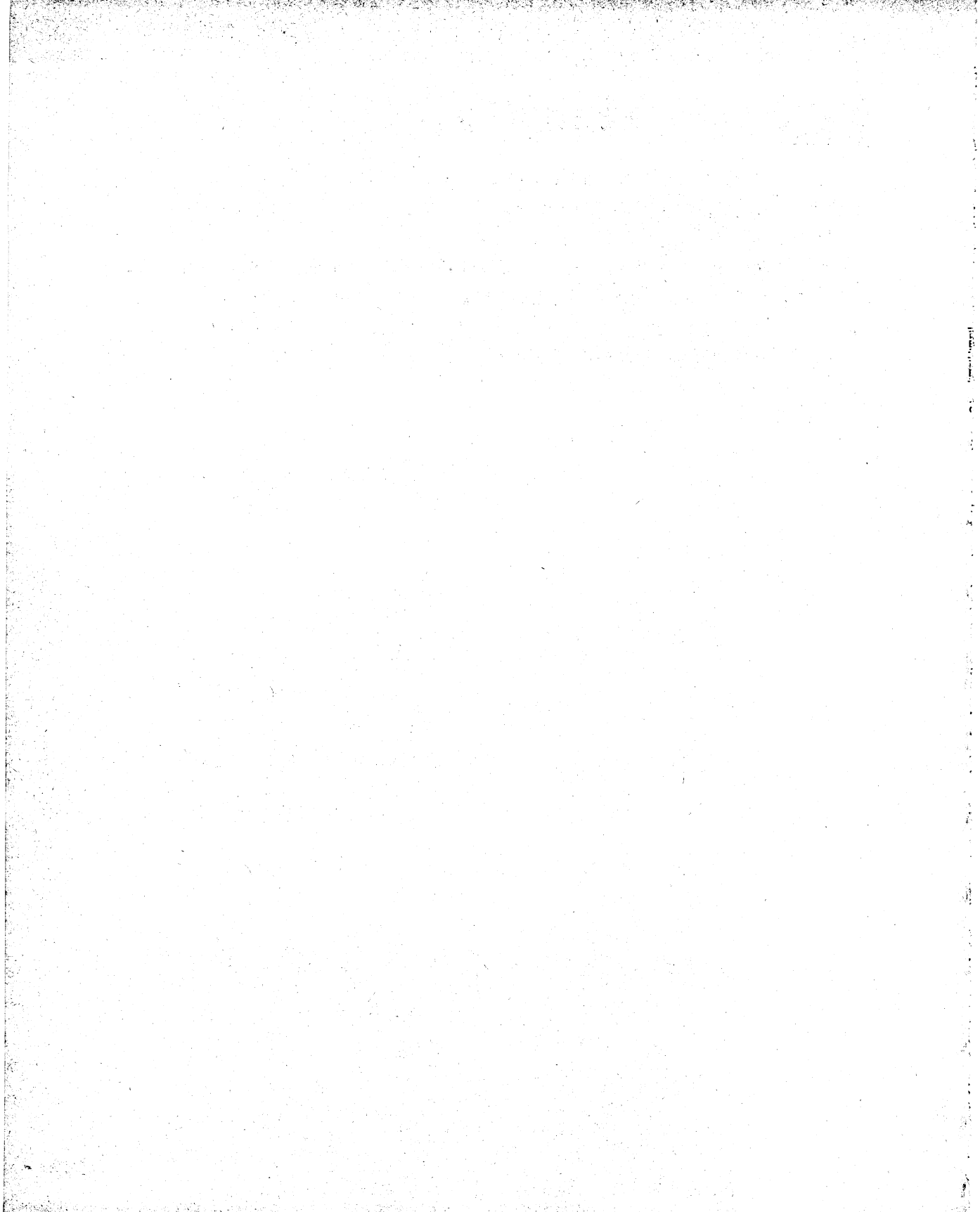
「グローバル変数はどこからもアクセスできて便利だよね」…確かにそうですね。「数のスコープがどこから始まってどこで終わって…」なんて気にする必要がありません。

しかし、だからといって全部をグローバル変数にしてしまったらどういうことが起きるでしょうか。たとえばKAGで使われている変数の数はざっと1000個以上あります。中には同じ名前の変数もたくさんあって、これをすべてグローバル変数にしてしまったら、同じ名前がぶつかってしまい、大変なことになります。

もちろん、グローバル変数はグローバル変数なりに便利なところがありますが、どこからも見えるという必要がないならば、ローカル変数にするべきである、というのはTJSに限らず多くの言語で言われていることです。グローバル変数とローカル変数、それぞれをうまく使い分けるのがコツです。

8-2 関数のスコープ

スコープに影響されるのは変数だけではありません。関数のスコープも (変数のそれとは若干違いますが) あります。これらはあとで「クラス」の機能を説明するときに一緒に説明することになります。

KAGのTJSスクリプトを見ると、いちばん外側のブロックの前に「class」と書いてあって、そのブロックの中に変数や関数がたくさん書いてあると思いますが、これが「クラス」です。このようなクラスを使わない場合は、関数はすべてグローバル位置に宣言されると覚えておけば、今はよいでしょう。

第9章 while文

「決められた動作を繰り返し実行したいというときのために、TJSはいくつかの繰り返し(ループ)の方法を用意しています。ここで紹介するのはそのうちのひとつのwhile文と呼ばれるものです。

■ ～～している間中

TJSにはいくつか繰り返し(ループ)を作る構文がありますが、while文はそのうちのもっとも簡単な形のものです。

とりあえず例を見て見ましょう。「UpdateConfig.tjs」には、以下のようなwhile文があります。

```
while(oldconfig[i] != end_mark && i < oldconfig.count)
{
        content += oldconfig[i] + "¥n";
        i++;
}
```

while文はこのように、まず「**while**」と書き、その ( ) 内に条件式を書きます。その次には文またはブロックが来ます。上記の例ではブロックが書いてありますね。この条件式が“真”の間中、ずっとこのブロックを実行するという意味になります。

でも、「“真”の間中」というのは少し誤解を招く表現かもしれません。while文の動作を詳しく見てみましょう。

①まず条件式を評価し､これが“偽”ならば文やブロックを実行せずに次に進む。
②文やブロックを実行する。
③条件式を評価し、“偽”ならば次に進み、“真”ならば②まで戻る。

つまり、文やブロックを実行している途中にたとえ条件式が“偽”になっても、このループを抜けることはありません。常に条件式をチェックしているわけではなくて、文やブロックを一回実行するごとにチェックしています。

また、while文を実行しようとする時点ですでに条件式が“偽”になっていた場合は、文やブロックは1回も実行されないので、注意が必要です (後述のdo～while文だと違います)。

「ブロック」ではなく「文」を使った例も挙げましょう。同じ「UpdateConfig.tjs」にあるwhile文です。

```
while(newconfig[i] != end_mark && i < newconfig.count)
i++;
```

この場合は、「newconfig[i] != end_mark && i < newconfig.count」という条件が“真”である限り「i++」を実行するという意味です(ちなみに「i++」とは「iに1を加算する」という意味です)。

■ break文

繰り返すのをブロックの途中でやめたい(ループの途中で抜け出したい)というときに使うのが「break文」と呼ばれるものです。

TJS2リファレンスにある以下の例を見て見ましょう。

```
while(true) // break が実行されない限りは無限ループ
 {
  if(func()) break; // func() が真ならばwhile終了
  func2(); // ↑で break が実行されなければここにくる
 }
 // break が実行されると、ここにくる
```

このwhile文は、条件式に「true」が指定されています。条件が常に“真”、つまり、ループを抜け出そうとしなければ永遠に繰り返す、「無限ループ」と呼ばれるものです(無限ループは吉里吉里の実行を止めてしまうので、書いてはいけません)。

ただ、この例ではループを抜け出すためにbreak文が書いてあります。break文が実行されるとブロックの残りの実行をせずに、すぐにwhile文の次に実行を移します。なにか条件によってループを抜け出したい場合がほとんどですから、break文は必ずといってよいほどif文やそれに類するもので使われます。

■ continue文

繰り返しの最初に戻りたいという場合に使うのが「continue文」と呼ばれるものです。

例を見てみます。

```
var i = 0;
while(i < 5)
 {
```

```
  func0();
if(func1()) continue;
// func1() が true ならば、i < 5 の条件式チェックがされ、真ならば
      // またブロックの先頭から実行される
      func2();
      i++;
  }
```

continue文は、実行されるとブロックのそれよりも後を実行せずに、条件チェックに戻ります。その条件チェックが“真”ならば、またブロックの最初から実行されますし、“偽”ならばwhileループから抜けることになります。

continue文もbreak文同様、if文やそれに類するもので使われます。

■ do～while文

章の始めでもちょっとお話しましたが、「do～while文」というのがあります。構文は以下のようになります。

```
do
   文またはブロック
while(条件式);
```

これは最初に条件チェックを行なわないことを除けば、while文と同じです。「do」の時点で条件が“偽”でも、文またはブロックが必ず1回は実行されます。

第10章

for文

for文も繰り返しを行なう場合に使う文ですが、while文やdo～while文より複雑なループを作ることができます。

■ for文の使い方

while文やdo～while文もfor文と同じく、繰り返しを行なうための文ですが、for文はもうちょっと高機能なものです。

いちばんオーソドックスな例を見てみましょう。「HistoryLayer.tjs」の「dispInit」という関数には以下のようなfor文があります。

```
for(i= 0; i < dataLines; i++)
        drawLine(i);
```


forの次の（ ）の中には「;」(セミコロン)で区切られた３つの部分があるのが分かるでしょうか。前から順に、「第１節」「第２節」「第３節」と呼んでいて、それぞれ役割が決まっています。

第１節は、ループに入る前に実行する式を指定します。この例では、変数「I」に「0」を代入しています。

第２節には条件式を指定します。「**while**」と同じで、ここの条件式が“真”であるあいだ中、文やブロックを繰り返し実行します。この例では、「i < dataLines」という条件が“真”である間実行されます。

第３節には、文やブロックの実行の１回ごとに実行する式を指定します。この例では、「i++」、つまり「i」に「1」を足しています。

ところで、上で例に挙げたfor文はどのように動作するのでしょうか。

変数「i」は最初に「0」になります。次に「i < dataLines」という条件がチェックされます。「i」が「dataLines」よりも小さくない場合は、文またはブロックは１回も実行されません。「i」が「dataLines」よりも小さければ「drawLine(i);」が実行されます。

１回、文またはブロックが実行された後は「i++」が実行されます。「i」に「1」が加算されます。そしてまた条件がチェックされ、「i」が「drawLine」よりも小さい場合はずっと「drawLine(i)」と「i++」が実行されることになります。

たとえば、「dataLines」が「4」の場合は、「i」は「0」「1」「2」「3」と変化しながら「drawLine(i)」が4回実行されます。「5」ならば、「0」「1」「2」「3」「4」と変化しながら5回実行されます。つまり、「dataLines」で指定した回数だけ、「drawLine(i)」が実行されることになり、しかも「i」は「0」から順に1ずつ繰り上がってます。

for文は、このように何かを指定回数ぶんだけ実行したいというときによく使われるものです。使い慣れないうちは、「節のどれがどういう機能だったかな？」と分からなくなってしまいがちですが、そのうちにwhile文よりも書くのが楽になってくるでしょう。

上記の例はwhile文を使えば、以下のように書くこともできます。

```
i = 0;
while(i < dataLines)
{
        drawLine(i);
        i ++;
}
```

■ continue文

for文をwhile文に書き直した例をお見せしましたが、for文中のcontinue文はwhile文中のそれと違うので、注意が必要です。

while文の「continue」は、ブロックの残りを実行せずに条件判断に戻るものでしたが、for文の「continue」は、第3節を実行してから条件判断に戻ります。

例を見てみましょう。「MessageLayer.tjs」の「internalAssign」という関数には、以下のようなfor文があります。

```
for(var i = sl.count-1; i>=0; i--)
{
        if(sl[i] === void) continue;
        (略)
}
```

continueが実行されると、「i--」が実行されて、「i>=0」という条件式がチェックされ、もし条件が真ならば、再びこのブロックが実行されます。

このようにfor文におけるcontinue文は、「文やブロックの実行の1回ごとに実行する式」を忘れずに実行してくれます。

■ 第1節のvar

第1節に「var」が書いてあるfor文を見掛けると思います。たとえば、「Utils.tjs」の「number_format」関数には以下のようなfor文があります。

```
for(var i = 0; i < n_len; i++)
{
        var digit = n[i];
        if(digit >= '0' && digit <= '9') n_digits ++;
        else if(digit == '.' || digit == 'e') break;
}
```

「var」をこのように第1節に書くと、この時点で変数を宣言*できます。この変数のスコープはfor文の終わりまでです。つまり、for文が終わった時点でこの変数「i」は使えなくなります。これはループだけに用いる変数を使うのに便利です。

＊宣言
どのような変数を使うかあらかじめ決定すること。

第11章

switch文

switch文は、値それぞれについて場合分けした処理を行なうときに便利です。

■ switch文の使い方

switch文は、ある値があって、その値がどの値と同じ時にどのような処理をするかを書き連ねることができるものです。

例を見てみましょう。「Utils.tjs」に「str2num」関数がありますが、そこにあるswitch文です。

```
switch(ch)
{
case "0": res+="0";break;
case "1": res+="1";break;

          (略)

case "ー": res+="-"; break;
case "ー": res+="-";break;
default: res+=ch; break;
}
```

switchの後の()の中には式を書きます。この式の結果が、「case」の後に書かれたものと一致する場合、その「case」のところの文が実行される仕組みです。たとえば、「ch」が"0"ならば、「res+="0"; break;」という文が実行されます。

「default:」は、どれにも当てはまらなかった場合に実行されるものです。

■ switch文におけるbreak

上記の例で、いちいちbreak文が書いてあるのに注目してください。break文は、ループで用いる「break」と同じように、この文が書かれた時点でswitch文を抜けます。break文を書くとswitch文を抜けるということは、break文を書かなかった場合は抜けないということです。その場合は、次に書かれた「case」の内容を実行してしまうので、注意が必要です。

オブジェクト

TJS2はオブジェクト指向的な考え方でプログラミングができる言語です。この章では、「オブジェクト」の概念や使い方について説明します。

12-1 オブジェクトとは?

「データ型」のところで、TJSの扱えるデータ型には種類がいくつかあると話したと思います。ほかの言語を知っている方の中には、そこに挙げたデータ型の中にはいくつか足りない型、たとえば「配列型」が無いので、「配列型はどうなるんだ」などと思った方もいるかもしれません。

*オブジェクト
object

「オブジェクト*」は英語で「もの」という意味で、特に何か特定のものを具体的に示す単語ではありません。TJSにおけるオブジェクトも、なにか特定の「これ!」といったものではありません。実際オブジェクトと言われるものの種類はけっこう多くて、先に説明した関数もオブジェクトの一種ですし、後で説明するクラスもオブジェクトの一種であり、配列や辞書配列もそうです。

実際のところ、いろいろなデータのうち、文字列型でも数値でもオクテット列型でもvoidでもなければ、それはオブジェクト型で、TJSから見ればオブジェクトとして扱われるものなのです。

■ メンバ選択演算子

*メンバ
member

オブジェクトは、たいていメンバ*を持っています。「メンバ」とは、そのオブジェクトが持っている変数や関数や他のオブジェクトなどのことです。TJSは、そのメンバにアクセスするために「メンバ選択演算子」というものをもっています。

たとえば、“Hello world!” のところで、以下のような式を使いました。

```
System.inform("Hello world!")
```

この「System.inform」というところに注目してください。「System」の次に「.」(ドット)が書いてあって、次に「inform」と書いてありますね。この「.」(ドット)は、「**直接メンバ選択演算子**」といいます。左側に書いたオブジェクトの中の、右側に書いたメンバにアクセスするよ、という動作をする演算子です。この場合は「System」というオブジェクトの中の「inform」というメンバにアクセスする、という意味です。日本語の「の」と考えるとわかりやすいでしょう。「Systemのinform」なのです。

メンバ選択演算子にはもうひとつ、「間接メンバ選択演算子」というのもありますが、これは次の章の辞書配列のところで説明します。

12-2 オブジェクトの種類

TJSが扱えるオブジェクトについて紹介します。

■ グローバル・オブジェクト

TJSが持っているやや特殊なオブジェクトとして、「グローバル・オブジェクト*」というのがあります。グローバル・オブジェクトはグローバル位置に宣言されたもの、つまりグローバル変数や普通の関数などをもっているオブジェクトです。これには「**global**」でアクセスすることができます。

＊グローバル・オブジェクト
global object

たとえば、グローバル位置で以下のような変数を宣言したとします。

```
var foo = 5;
```

この変数「foo」はグローバル変数として宣言されました。実は、この時点でグローバル・オブジェクトは「foo」というメンバをもつことになります。

試しに、スクリプト・エディタで「**var** foo = 5;」と入力して実行してみてください。次に、コンソールで「**global**.foo」と入力してみてください。「5」と表示されるはずです。

逆に、「**global**.foo = 1」とコンソールに入力すれば、変数「foo」は「1」という値をもつことになります。コンソールに「foo」と入力してみてください。「1」と表示されたはずです。

また、今までたびたび使ってきた「System.inform」の「System」というのも「**global**」のメンバです。ですから、たとえば「System.inform("Hello world!");」を「**global**.System.inform("Hello world!");」としてもちゃんと動きます。

■ 関数とプロパティ

関数もオブジェクトとして扱われます。オブジェクトは変数に入れることができるので、変数に関数を入れてアクセスするようなことができます。

たとえば、以下のようなスクリプトを書くことができます。

```
function add(a, b) { return a+b; }
var foo = add;
System.inform(foo(2, 3));
```

この例では、変数fooに関数addを代入しています。すると、fooがあたかも

関数であるかのように使えるようになります。

「**プロパティ***」というのは、ここでは説明しませんが、一見変数のように見える関数のようなものです。変数のように値を設定したり、値を取り出したりできますが、そのときに関数のように特定のスクリプトを実行することができるものです。

*プロパティ
property
特性

■ クラスとオブジェクト

「クラス」は**14章**で詳しく説明しますが、オブジェクトを作るためのひな形のようなものです。どのような種類のオブジェクトを作るか、どんな変数があってどんな関数を持っているのか、を知っているのが「クラス」です。

実は、クラスそのものもオブジェクトの一種です。C++などの言語ではクラスは「型」として扱われ、オブジェクトとは厳しく区別されるのですが、TJSでは同じ種類のものとして扱います。クラスを変数に入れてほかの関数に渡し、そのクラスからオブジェクトを作るといった使い方もできます。

■ 配列と辞書配列

TJSにおいては、配列もオブジェクトの一種です。配列が番号で要素を管理するのに対し、名前で要素を管理する「辞書配列」というのもあります。どちらも次の章で詳しく解説します。

■ 文字列に対する操作

オブジェクトに対してメンバ選択演算子を使うとそのメンバにアクセスすることができると話しました。しかし実は、文字列や文字列の入っている変数に対しても、メンバ選択演算子を使うことができます。文字列はTJSにおいてはオブジェクトではないのですが、擬似的にまるでオブジェクトであるかのようにメンバ選択演算子を使うことができます。

たとえば、文字列の長さを表わす「length」というメンバがあります。試しにコンソールから「"abc".length」と入力してみましょう。文字列「"abc"」は3文字の長さがありますから、「3」と表示されたと思います。もちろん、変数に対しても同じで、たとえば変数「hoge」に「"abc"」という文字列が入っていれば、「hoge.length」は「3」になります。

文字列に対する操作はいくつか便利なものがあります。「TJS2リファレンス」を見てみてください。

第13章

配列

大量のデータを使いたいときに、その数だけ変数を分作っていたら大変です。効率良く大量のデータを処理したいと思ったら、「配列」を使いましょう。

13-1 配列

「配列」は、大量のデータを、通し番号をつけて管理できるものです。たとえば、1000個のものを記憶したいからと言って変数を1000個宣言していたら大変なことになります。配列を使えば、配列の中にその1000個を入れておくことができます。

TJSでは配列もオブジェクト*です。配列は「Array」というクラス*から作られるオブジェクトです。「クラス」は、簡単にいうとオブジェクトを作るためのひな形なのですが、これは次の章で詳しく説明します。

*オブジェクト
object

*クラス
class

■ 配列の作成

配列は使う前に「配列オブジェクト」を作らなければなりません。配列オブジェクトを作るには、new 演算子を使って、以下のようにします。

```
var ar = new Array();
```

*筆者注
new Array(); の代わりに []; としても配列オブジェクトを作成できます。

このように「new Array();」とすることで配列オブジェクトを作ることができます。上記の例では、変数arにそのオブジェクトを入れています。

■ 配列のアクセス

配列の各要素は、「0」から始まる通し番号で管理されます。この通し番号を添え字と呼びます。添え字を指定するには [ ] 演算子(間接メンバ選択演算子)を使います。

たとえば、以下のように使います。

```
ar[0] = "zero";
ar[1] = "one";
ar[2] = "two";
```

この例では、0番目に「"zero"」を、1番目に「"one"」を、2番目に「"two"」を入れています。この状態でたとえば「System.inform(ar[0])」とすれば「zero」と表示されることになります。

何も値が代入されていない要素は「void」であると見なされます。

■ 配列の大きさ

配列の要素数(配列の大きさ)は count プロパティで参照できます(プロパティはオブジェクトのメンバで、変数のようなものです)。

たとえば、上記の例の要素数を確かめたかったら、コンソールか何かで「ar.count」としてみましょう。ar 変数に入っている配列オブジェクトのもっている要素数である「3」と表示されるはずです。「ar.count」になにか値を入れても要素数を変えることができます。

配列の大きさは、その配列に代入するときに使った添え字のいちばん大きなものになります。上記で言えば「ar[2]」がいちばん大きいですから、要素数は「3」になります(添え字は「0」から始まっていることに注意してください)。

配列の大きさは好きなだけ大きくできますが、そのぶん、大きなメモリが必要です。

13-2 辞書配列

TJS2は「辞書配列」と呼ばれるものもサポートします。辞書配列も配列の一種ですが、こちらは通し番号で管理するのではなくて、「名前」で管理します。名前に結び付けられた値を取り出す様子から「連想配列」などと呼ばれることもあります。

■ 辞書配列の作成

辞書配列を作るには、普通の配列と同じようにnew演算子を使い、以下のようにします。

*筆者注
new Dictionary(); の代わりに %[]; としても配列オブジェクトを作成できます。

```
var dic = new Dictionary();
```

■ 辞書配列のアクセス

配列の場合は、要素のアクセスに [ ] 演算子を指定しました。辞書配列の場合も同じです。

```
dic['zero'] = 0;
dic['one'] = 1;
dic['two'] = 2;
```

これは「'zero'」という名前の要素(zeroという名前のメンバ)に「0」を、「'one'」という名前の要素に「1」を、「'two'」という名前の要素に「2」を代入しています。この状態で「System.inform(dic['zero']);」とすれば、「0」と表示されるはずです。

名前は重複することはありません。たとえば、この状態で「dic['zero'] = "ぜろ";」とすれば、「dic['zero']」という要素の内容は「"ぜろ"」に置き換わります。「'zero'」という名前の要素が2つになるわけではありません。

要素を削除するにはdelete演算子を使います。たとえば「delete dic['two'];」とすれば、「'two'」という名前の要素を削除できます。

初めて使われた名前の要素には「void」が入っているものと見なされます。

■ KAGの変数と辞書配列

さて、前にKAGの変数は辞書配列であると話しました。実は「f」や「sf」は辞書配列なのです。たとえばゲーム変数「f.hoge」は、「f」という辞書配列の「hoge」という名前をもった要素にアクセスしていたのでした。

「あれ？辞書配列の要素には [ ] 演算子でアクセスするんじゃないの？」と思うかもしれません。「.」(ドット)は直接メンバ選択演算子、[ ] は間接メンバ選択演算子です。直接か間接かの違いはあっても、両者は同じ仲間の演算子なのです。「直接メンバ選択演算子」は、「.」の右側にメンバの名前を「直接的に」書いてアクセスします。それに対し、「間接メンバ選択演算子」は、[ ] の中にメンバの名前を文字列として「間接的に」書いてアクセスします。

「直接メンバ選択演算子」を使うと「f.hoge」と書くことができますが、「間接メンバ選択演算子」を使うと「f['hoge']」とかけます。両者はまったく同じ意味になります。

「間接メンバ選択演算子」の利点は、[ ] 内に変数を指定できるということです。たとえば、「var name="hoge";」として、「name」という変数には「"hoge"」という文字列が入っているとします。そこで「f[name]」と書けば、「f["hoge"]」と同じ意味、つまり「f.hoge」にアクセスできるのです。

13-3 配列のコピー

配列や辞書配列をコピーしたい場合には、どのようにすればよいでしょうか。配列に限らないことなのですが、TJSはオブジェクトを「参照」という方法で扱います。

■ 参照とは？

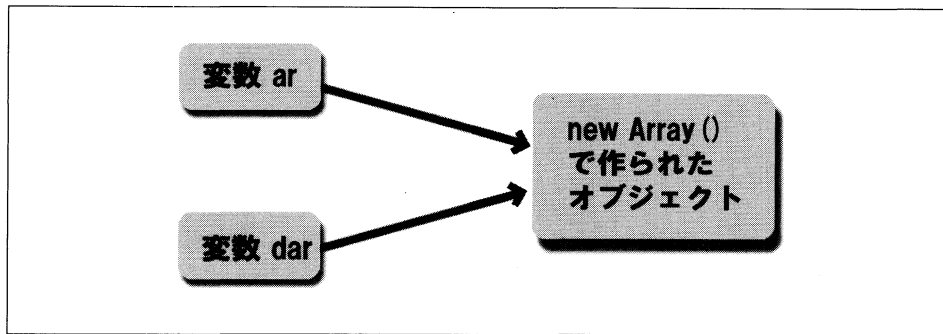
配列をコピーしたいとして、たとえば以下のようにしてみましょう。変数「ar」には配列オブジェクトを入れて、それを変数「dar」にコピーしたいとします。

```
var ar = new Array();
ar[0] = "zero"; ar[1] = "one"; ar[2] = "two";
var dar = ar;
```

はたしてこれで良いのでしょうか。結論から言うと、これでは駄目なのです。
試しにこのスクリプトをスクリプト・エディタから実行してください。その後、コンソールから「ar[0]」の内容を確かめてください。「"zero"」と表示されたと思います。では、「dar[0] = "ぜろ"」とコンソールから入力し、「dar[0]」に「"ぜろ"」を入れます。その後、またコンソールから「ar[0]」の内容をみてみてください。「"ぜろ"」になっていると思います。つまり、コピー先の内容を書き換えたはずが、コピー元の内容も書き換わってしまっているのです。

これは、TJSが「オブジェクト」を「=」(イコール)演算子や関数の引数への引き渡しなどでほかの変数などに入れるときに、「コピー」ではなくて「参照」を増やすという方法で行なうからです。これは配列に限らず、TJSがオブジェクトとして扱うもの全部に言えます。

「参照」はC言語などの言語を使っている方には「ポインタ」の方が分かりやすいかもしれません。実体となるオブジェクトがあって、それを「指し示す」、つまり参照する変数が増えるだけなのです。
上記の例でいえば、「new Array()」で作ったオブジェクトの実体はまず「ar」という変数から参照されています。しかし、そのあと「dar」という変数からも参照されます。結局、「ar」も「dar」も同じオブジェクトを指し示していることになるのです。

＊C言語
米国のベル研究所で開発された高級言語。UNIXにおける標準開発言語。

オブジェクトは「参照」されている

■ 配列のコピー

配列を本当にコピーしたい場合は、配列の「assign」というメンバ関数を使う必要があります。

上記の「dar = ar;」は以下のようになります。

```
var dar = new Array();
dar.assign(ar);
```

まず、「dar」に新しい配列オブジェクトを作って入れます。そのあと、その新しい配列のメンバ関数である「assign」を呼び出し、「ar」の内容をコピーします。これで、「dar」に「ar」の内容がすべてコピーされるのです。

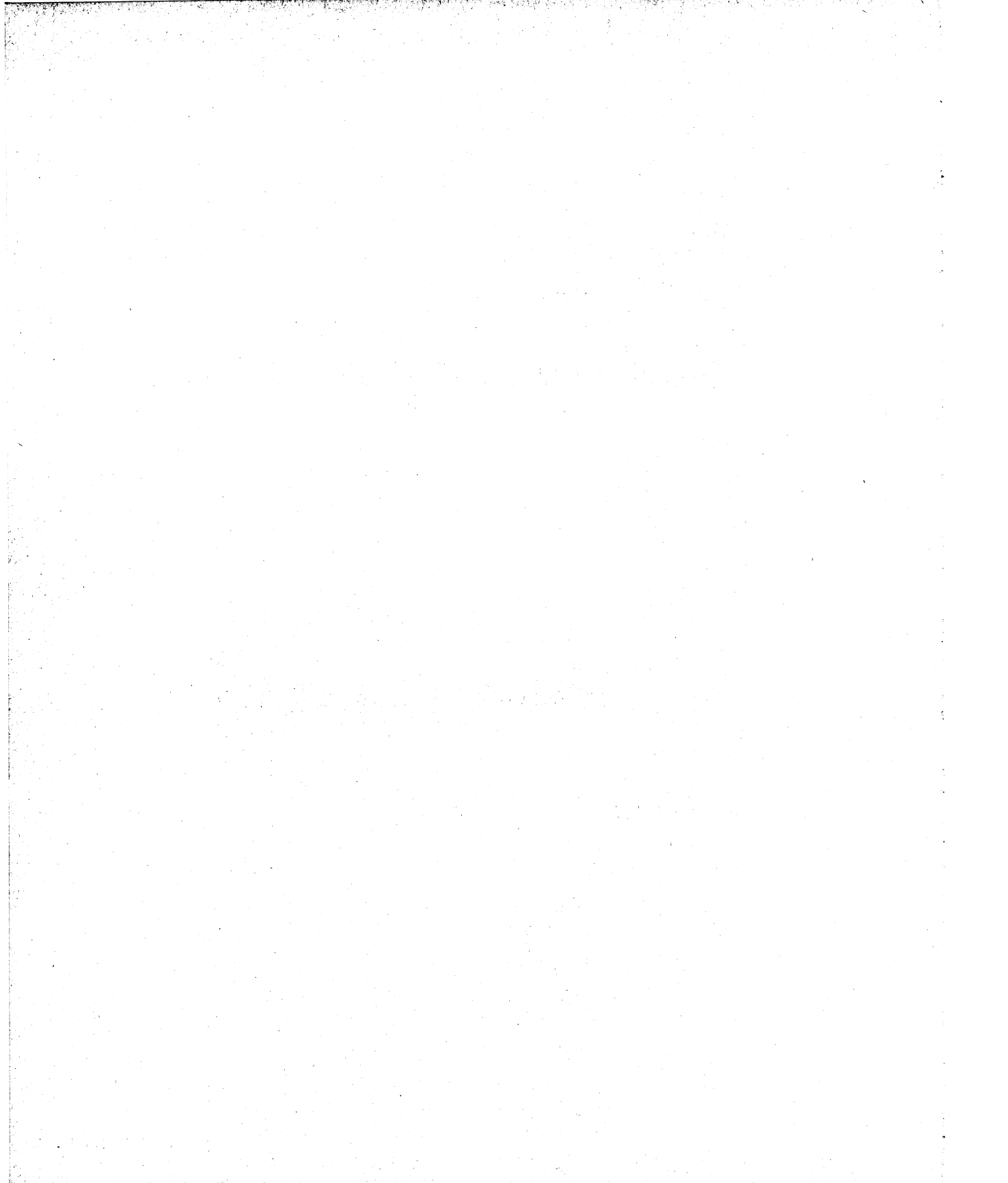
■ 辞書配列のコピー

辞書配列のコピーはちょっとやっかいです。辞書配列オブジェクトは作られた直後はメンバを何も持っていないからです。「assign」というメンバさえありませんので、上記の配列と同じ方法は使えません。

どうするのかといえば、たとえば「ddic」という変数に「dic」という辞書配列をコピーしたければ、

```
var ddic = new Dictionary();
(Dictionary.assign incontextof ddic)(dic);
```

と書く必要があります。

ややこしいですが、「Dictionary.assign」という関数を、コピー先である「ddic」というオブジェクトに対して呼び出しています。引数にはコピー元の辞書配列を指定します。とりあえずはこのように書くものだ、と覚えておけばよいと思います。

第14章

クラス

クラスはオブジェクトを作るための、いわば「ひな形」です。オブジェクトの役割や性質を決める「クラス」について説明します。

14-1 クラスとは？

TJSには多くの種類の「オブジェクト」があるという話をしました。しかし、その「種類」はどのように決定されるのでしょうか。

そのカギを握るのが、「クラス」と呼ばれるものです。クラスはオブジェクトの「ひな形」のようなものです。TJSでは通常、オブジェクトを作るときにこの「クラス」を用いて作ります。オブジェクトの「種類」やその「性質」を決めるものが、クラスなのです。

KAGのTJSスクリプトを見ると「**class**」、つまり「クラス」を表わす表記がよく出てくると思います。事実、KAGではほとんどの記述が**class**に続くブロック内に書いてあります。KAGを理解するにはこの「クラス」を知り、オブジェクトとどう関わりがあるのかも知ることが重要です。

■ クラスを見てみよう

比較的独立していて分かりやすい例が「YesNoDialog.tjs」にあるので、見てみましょう。ここには「YesNoDialogWindow」というクラスが定義されています。このクラスは「はい」か「いいえ」をユーザに選択させるためのウィンドウを作るためのひな形です。KAGでは終了しようとするときに「終了しますか？」と表示されますが、あれを司っているのがこのクラスから作られたオブジェクトなのです。

このクラスの構造は、大まかに以下のようになっています(一部省略してあります)。

```
class YesNoDialogWindow extends Window
{
        var yesButton;
        var noButton;
        var result = false;
        function YesNoDialogWindow(message, cap)
        {
                super.Window();
                (略)
```

```
    }
    function finalize()
    {
        super.finalize(...);
    }
    function action(ev)
    {
        (略)
    }
    function onKeyDown(key, shift)
    {
        super.onKeyDown(...);
        (略)
    }
}
```

クラスは「**class**」から始まるブロックです。この中に、このクラスから作られるオブジェクトがどのようなメンバをもつか、すなわちどのような性質をもつかを記述することになります。

14-2 クラス内に記述するもの

classに続くブロック中には、変数や関数を書くことができます。

■ メンバ変数

クラスでは、オブジェクトが作られるときに、そのオブジェクトがあらかじめもつ変数、「メンバ変数」を宣言*できます。記述の方法は普通の変数と同じで、「var」を使って宣言します。

*宣言
どのような変数を使うかあらかじめ決定すること。

■ メンバ関数(メソッド)

オブジェクトがもつ関数を宣言することもできます。記述の方法は普通の関数と同じで、functionを使って宣言します。「メンバ関数」のことをよく「メソッド*」とも呼びます。

*メソッド
method

メンバ関数の中では、オブジェクトのメンバ変数にそのままアクセスすることができます。オブジェクトのメンバ変数には「.」(ドット)演算子を使う、と書きましたが、自分自身のオブジェクトのメンバ変数にアクセスする限りは、その必要はありません。

たとえばメンバ関数「action」には「result = **true;**」や「result = **false;**」という記述がありますが、このresultはメンバ変数ですね。クラス内で「**var**」を使って宣言されている変数です。

*筆者注
実際には「プロパティ」というのも記述できますがここでは説明しません。

14-3 オブジェクトの作成と削除

クラスそれ自体は単なるひな形なので、実際にはこれからオブジェクトを作らなければなりません。オブジェクトを作るには、今までにも出てきましたが、「new」を使います。クラス「YesNoDialogWindow」からオブジェクトを作っている部分が、「YesNoDialog.tjs」にあるので、見てみましょう。

この関数「askYesNo」は、先ほどのクラスを使って指定されたメッセージを表示し、ユーザーに「はい」か「いいえ」を選んでもらう関数です。

```
function askYesNo(message, caption = "確認")
{
        var win = new YesNoDialogWindow(message, caption);
        win.showModal();
        var res = win.result;
        invalidate win;
        return res;
}
```

「new」は演算子で、「new」の次に書いたクラスから、オブジェクトを作る、という意味になります。ここでは「new YesNoDialogWindow」としていますね。その次に ( ) があって、この中には「コンストラクタ*」(後で説明します)に渡す引数を指定します。

＊コンストラクタ
constructor

作ったオブジェクトは変数winに入ります。「win.showModal()」では、作ったオブジェクトの「showModal」という関数を呼び出しています。

「var res = win.result;」では、作成したオブジェクトのメンバ変数である「result」を変数resに入れています。このメンバ変数は、先ほどのクラス内で「var」を使って宣言したものです。

■ コンストラクタとは？

「コンストラクタ」は日本語では「構築子」と呼ばれるもので、特別なメンバ関数です。オブジェクトが作られるときには、必ずこの関数が呼ばれます。

コンストラクタは「クラス名と同じ名前になる」という特別なルールがあります。先ほどのクラス「YesNoDialogWindow」にも「YesNoDialogWindow」という関数がありますが、これがコンストラクタです。

new演算子にはコンストラクタに渡す引数を指定する、といいました。メンバ関数「YesNoDialogWindow」には「message」と「cap」という2つの引数があります。先ほどの関数「askYesNo」では、この引数を「new」で指定したのです。

■ オブジェクトを削除する

オブジェクトは作ったら「削除」しなければなりません。幸い、TJSは作ったオブジェクトは自動的に削除されます(例外もあります)。「配列」の章で「参照」というお話をしました。この「参照」がすべてなくなる、つまりオブジェクトがどこからも参照されなくなると、作られたオブジェクトは自動的に「削除」されます。

■ 削除と無効化

オブジェクトが削除される前に、TJSでは「無効化」という段階を踏みます。「無効化」とは、「そのオブジェクトは使えない」、つまり「そのオブジェクトは無効」とすることです。

無効化は削除の前に必ず起こるもので、削除が自動的に起こる場合は無効化も自動的に起こります。しかし、意図的にオブジェクトを無効化させることもできます。それがinvalidate演算子です。

先ほどの関数askYesNoにもありました。「invalidate win;」として、変数winが参照しているオブジェクトを無効化したのです。

無効化するとそのオブジェクトはもう使えなくなります。あとはそのオブジェクトはTJSによって自動的に削除されるのを待つだけとなります。

■ 削除のタイミング

参照がすべてなくなるとオブジェクトは自動的に削除されるといいましたが、これはちょっと嘘なのです。実は参照がすべてなくなってもオブジェクトが即、削除される保証はありません。TJSは参照がすべてなくなってもオブジェクトをすぐには削除しないのです*。削除のタイミングが分からない、ということになります。

これは実行効率を上げるための工夫なのですが、これが災いすることがあります。自動的に削除が起こる前に自動的に無効化が起こるのですが、削除のタイミングが分からないということは、この無効化のタイミングでさえ、予測できないものとなるのです。

無効化は削除と違い、後述のfinalizeメソッドを呼び出すことがあり、この関数が呼ばれるタイミングが予測できないというのが問題になる場合があります。そのため、明示的にinvalidate演算子を使っておけば、確実にその時点で

*筆者注
オブジェクトをすぐに削除しないことで実行効率を上げる仕組みのことを「ガベージ・コレクション」といいます。

オブジェクトを無効化し、finalizeメソッドを呼んでおける、ということになるのです。

invalidate演算子を使うか否かは難しいところですが、一応、「作ったオブジェクトは必ずinvalidateする」と覚えておけばよいでしょう。ただし、「Array」や「Dictionary」クラス、つまり配列や辞書配列などや、正規表現を使うための「RegExp」クラス、「例外*」の際に使う「Exception」クラスなどは、「invalidate」を使わず、無効化もTJS側に任せてしまうことがよくあります。これらは無効化時には特にこれといって何もしないので「invalidate」を使わなくても特に問題がないのです。それ以外は、普通は「invalidate」を使います。

*例外
通常の処理中では考えられない、例外的なエラーなどの処理。

■ finalizeメソッド

オブジェクトが無効化されるときに呼ばれるのがこの「finalize」というメンバ関数、「finalizeメソッド」です。オブジェクトが無効化、つまり「このオブジェクトは使えない(使わない)」とされるときに呼ばれる関数ですから、ここには「後処理」、つまりオブジェクトの後片付けの処理を書くことになります。

「finalizeメソッド」は、「コンストラクタ」とは反対の動作をするという意味で、「デストラクタ」(消滅子)と呼ばれることもあります。

■ オブジェクトを作らないクラス

「クラスはオブジェクトのひな形である」と話しましたが、例外があって、ひな形として機能しないクラスもあります。

クラスもオブジェクトの一種なのですが、このクラスというオブジェクトの特徴として、クラス内で宣言したメンバ関数をすべてメンバとして持っているということが挙げられます。通常はメンバ関数は、そのクラスからnew演算子でオブジェクトが作られることを想定して書きますが、そうでなく、クラス内に単に関数をいくつか書けば、その関数群をクラスという形でまとめることができるのです。

「吉里吉里」が提供しているいくつかのクラスがそれで、たとえば今までに何度か「System.inform」の形で紹介した「System」というのも、実はクラスなのです。

ただし、「System」はクラスだといっても「new System()」のような使い方はしません。「System」というクラスは「吉里吉里」本体や、「吉里吉里」が実行されている環境に関する情報を取得したり、設定したりするためのクラスで、単にこの機能に属する関数をクラスの形にまとめただけのものなのです。

14-4 継承

クラスに関する非常に重要な概念として「継承」というものがあります。クラスはオブジェクトの「種類」や「性質」を決めるものですが、継承によって引き継がれるのは、この「種類」や「性質」です。

私たちの世界でもこの継承という概念は多くに当てはまります。

私たち人間を考えてみましょう。私たち「人間」は「ほ乳類」ですね。「人間」は、子が母の母乳で育つという「ほ乳類」の性質を「継承」しています。その「ほ乳類」は「動物」の「よく動いて活動する」という性質を「継承」しています。その「動物」は「生きている」という「生物」の性質を「継承」しています。私たち人間は、生物であり、動物であり、ほ乳類であり、人間であり、継承元の性質をすべて引き継いでいるのです。

また、「生物→動物→ほ乳類→人間」という順では、より「抽象的」な「生物」から、より「具体的」な「人間」へと継承されていっているのが分かると思います。

TJSのクラスもほかのクラスの性質を「継承」することができます。たとえば、先ほどの「YesNoDialogWindow」というクラスは、「Window」というクラスを「継承」しています。Windowは「吉里吉里」が提供するクラスで、いわゆる「ウィンドウ」のひな形です。「YesNoDialogWindow」は、この「ウィンドウである」という性質を継承しつつ、「『はい』か『いいえ』かをユーザーに選択させるためのウィンドウ」という性質をもつひな形なのです。

■ 継承に関する用語

継承に関する用語はいくつかあります。

・継承元の(より抽象的な)クラスを「親クラス」あるいは「スーパークラス」と呼びます。
・継承した(より具体的な)クラスを「子クラス」あるいは「サブクラス」と呼びます。
・クラスを継承させることを「親クラスから子クラスを派生する」とも呼びます。

■ 継承のしかた

クラスをクラスから継承させるには、「**extends**」を使います。先ほどのクラス「YesNoDialogWindow」の**class**宣言のところを見てください。

```
class YesNoDialogWindow extends Window
```

となっていますね。これは「YesNoDialogWindow」を「Window」から継承させる、つまり「Window」から「YesNoDialogWindow」を派生させています。

こうすることで、YesNoDialogWindowから作成されたオブジェクトは、Windowクラスのメンバもすべてもつことになります。

先ほどの関数「askYesNo」の中で、「win.showModal();」というのがありました。「showModal」は「YesNoDialogWindow」内で定義されていないメンバ関数なのでいったいどうしたものか、と思った方もいるかと思いますが、これは実はWindowクラスに属するメンバ関数で、「ウィンドウを表示し、そのウィンドウが閉じるまで、ほかのウィンドウへのアクセスを禁止する」という機能をもったメンバ関数です。

■ 継承とコンストラクタ

先ほどのクラス「YesNoDialogWindow」のコンストラクタ、つまり関数「YesNoDialogWindow」を見てみてください。

```
                super.Window();
```

とありますね。この**super**は「スーパークラス」、つまり「親クラス」を表わすものです。

親クラスはWindowクラスですが、Windowクラスにもコンストラクタがあり、その名前はクラス名と同じ「Window」です。つまり、この「**super**.Window()」は、親クラスのコンストラクタであるWindowというメンバ関数を呼んでいるのです。

このように、クラスを継承したら、その親クラスのコンストラクタを、子クラス内のコンストラクタから呼ばなければなりません。

■ 継承とfinalize

finalizeメソッドも、親クラスのものを呼ばなければなりません。クラス「YesNoDialogWindow」の「finalize」にも以下の記述があります。

```
super.finalize(...);
```

ここで親クラスの「finalize」というメソッドを呼び出しています。関数の引数のところにある「...」(ドット３つ) は、この関数に渡された引数をそのまま別の関数に引き渡す、という意味です。実際には仕様として「finalize」には引数は何も渡されないのですが、慣用的にこのように書いています。

■ オーバーライド

継承に付随する、これまた重要な概念として「オーバーライド」というものがあります。

オーバーライドは「優先する」という意味で、この場合は「子クラスのメンバ関数が親クラスのメンバ関数に優先する」という意味です。

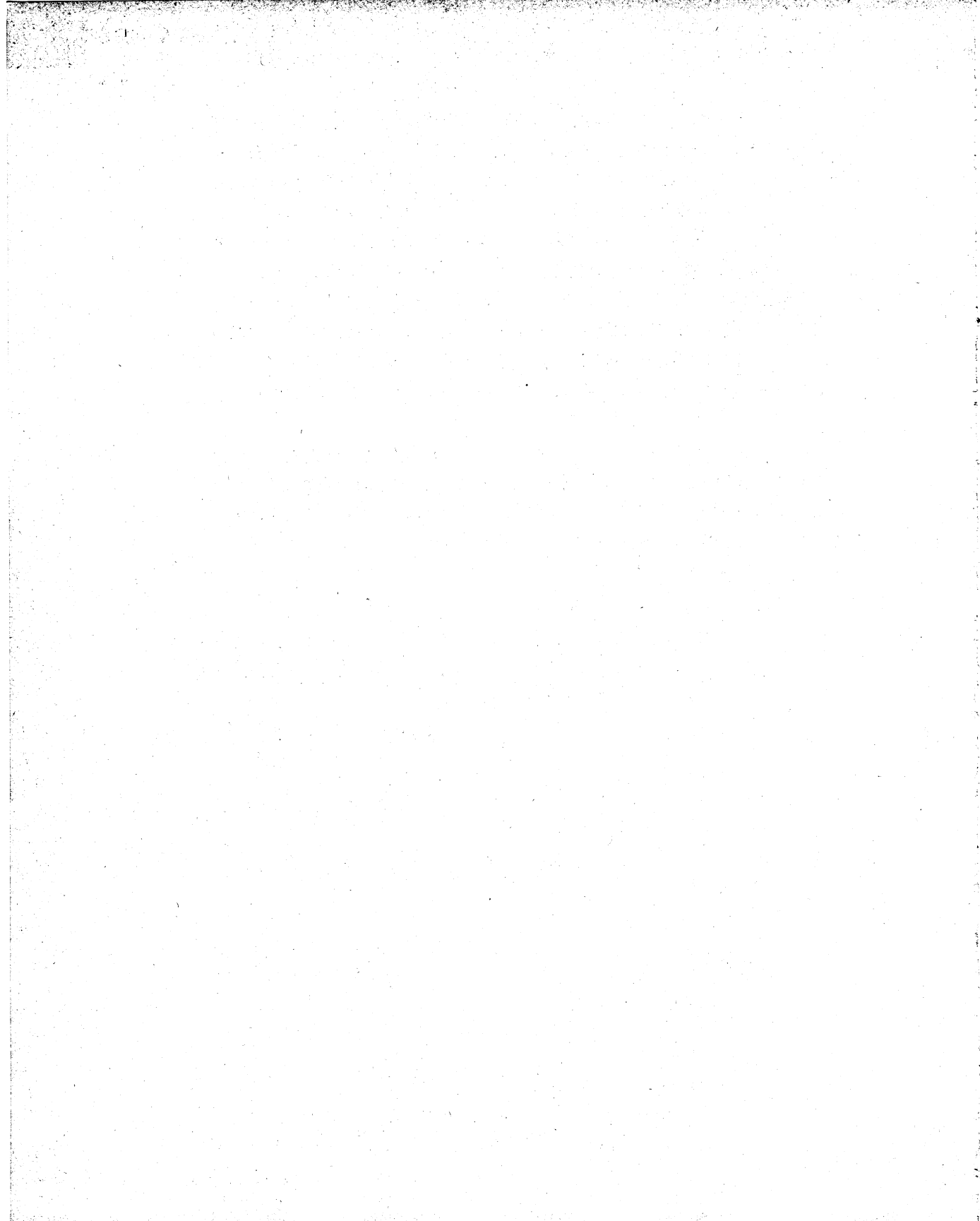
どういうことかというと、子クラスでは親クラスにあるメンバ関数と同名のメンバ関数を宣言できます。このとき、親クラスのメンバ関数は子クラスのメンバ関数に隠されてしまいます。

先ほどのクラス「YesNoDialogWindow」の「onKeyDown」という関数を見てみてください。同名の関数は「吉里吉里２」のリファレンスを見ても分かるとおり、親クラスのWindowクラスにもあります。このようにすると、作られたオブジェクトの「onKeyDown」というメンバ関数は、子クラスの「onKey-Down」になり、親クラスの「onKeyDown」ではなくなるのです。

隠された親クラスのメンバはどうなるんだ、ということになりますが、これは親クラスを表わす**super**でアクセスできます。クラス「YesNoDialogWin-dow」にも「**super**.onKeyDown(...);」という記述があります。これは親クラスの「onKeyDown」を呼んでいるのです。

そういえば「finalize」もそうですね。「finalize」も親クラスの「finalize」をオーバーライドする形で書き、その中で親クラスの同名のメソッドを呼びます。

オーバーライドは、親クラスのメソッドの性質を変えたいが、名前はそのままにしたい(あるいは名前をそのままにしなければならない)、という場合によく使います。よく使うのは「イベント」、つまり「吉里吉里」側から呼ばれるメンバ関数を記述するときで、「吉里吉里」が提供するクラスにはあらかじめ動作が決まったメンバ関数が定義されています。動作が決まった、といってもほとんどは「何もしない」という動作ですが、この動作を変えたいときに、そのメンバ関数をオーバーライドして動作を記述する、ということを行ないます。
先ほどの「onKeyDown」は「キーが押された」ときに呼ばれるメンバ関数ですが、クラス「YesNoDialogWindow」ではこのメンバ関数をオーバーライドして、キーが押されたときに特別な処理ができるようにしてあります。

オーバーライドできるのはメンバ関数だけで、TJSではメンバ変数はオーバーライドできません。子クラスで親クラスと同じ名前のメンバ変数を作ると、親クラスのメンバ変数と子クラスのメンバ変数が同じものとして扱われ、「super」を使ってもアクセスできなくなるので、注意が必要です。

KAGの構造と動作

この章では、KAGの構造と、KAGや吉里吉里がどのように動作するのかの概要をお話しします。

15-1 吉里吉里/KAGの構造と動作

KAGがどのような構造になっていて、どのような動作をするかについて、概要をお話しします。

■ Systemフォルダのファイル

KAGのSystemフォルダにあるファイルは、以下のようになっています

LineBreak.asd	行末クリック待ち記号用アニメーション定義ファイル。
PageBreak.asd	ページ末クリック待ち記号用アニメーション定義ファイル。
check.png	チェック・ボックスのチェック記号(「CheckBoxLayer.tjs」で使用)。
LineBreak.png	行末クリック待ち記号用画像。
LineBreak_a.png	行末クリック待ち記号用アニメーション・セル画像。
PageBreak.png	ページ末クリック待ち記号用画像。
PageBreak_a.png	ページ末クリック待ち記号用アニメーション・セル画像。
AnimationLayer.tjs	アニメーションの進行を管理するコンダクタ用クラス「AnimationConductor」とともに、アニメーションを行なう機能を持ったレイヤのクラスである「AnimationLayer」を定義する。
BGM.tjs	BGMを管理するクラスBGMと、それに必要な諸クラスを定義する。
ButtonLayer.tjs	ボタンとして動作するレイヤ・クラス「ButtonLayer」を定義する。
Config.tjs	KAGの設定ファイル。
CheckBoxLayer.tjs	チェック・ボックスとして動作するレイヤ・クラス「CheckBoxLayer」を定義する。
Conductor.tjs	KAGのシナリオやアニメーションの進行を管理するための「コンダクタ」の親クラス「BaseConductor」と、シナリオ管理用のコンダクタ・クラス「Conductor」を定義する。
DefaultMover.tjs	レイヤの自動移動の「動き」を管理するクラスである「LinearMover」と「SplineMover」クラスを定義する(moveタグで使用)。
EditLayer.tjs	単一行エディット(文字入力欄)用レイヤクラス「EditLayer」を定義する。
GraphicLayer.tjs	KAGの背景レイヤや前景レイヤの親クラスである「GraphicLayer」と、背景レイヤ用クラスである「BaseLayer」、前景レイヤ用クラスである「CharacterLayer」を定義する。
HistoryLayer.tjs	メッセージ履歴レイヤ用クラスを定義する。
Initialize.tjs	システムの初期化を行なうプログラム。「startup.tjs」はこのファイルを実行するだけのものである。
KAGLayer.tjs	KAGで用いられているレイヤクラスすべての親クラスである「KAGLayer」を定義する。
MainWindow.tjs	KAGのウィンドウを管理するクラスである「KAGWindow」を定義する。KAGの動作の多くがここに記述されている。
Menut.tjs	メニュー項目の定義を行なう。
MessageLayer.tjs	KAGのメッセージ・レイヤを管理するクラス「MessageLayer」を定義する。

Movie.tjs	動画再生を管理するクラス「Movie」を定義する。
Plugin.tjs	KAG用プラグインの親クラスである「KAGPlugin」を定義する。
SE.tjs	効果音を管理するクラス「SESoundBuffer」を定義する。
UpdateCponfig.tjs	「Config.tjs」を古いバージョンのものから引き継ぐためのプログラム。
Utilt.tjs	いくつかのユーティリティ関数を定義する(乱数発生、漢数字変換など)。
YesNoDialog.tjs	「はい」「いいえ」をユーザに選ばせるためのクラス「YesNoDialog-Window」とそれを使うための関数を定義する。

■ イベント駆動型

＊イベント駆動
イベントドリブン(event driven)とも呼ばれる

「吉里吉里」でプログラミングをするときには、「イベント駆動型」というプログラミングの仕方をする必要があります。イベント駆動型のプログラムは、普段は待ち状態にあります。そこで「キーが押された」だとか、「タイマーの時間切れが来た」だとか、「マウスが動いた」など、いろいろな「イベント」が発生します。それに反応する形でいろいろと動作の記述を行うのが「イベント駆動型」と呼ばれるものです。このようにイベントに反応して動作するプログラムの部分のことを、「イベント・ハンドラ」と呼びます。

「イベント駆動型」に対する言葉として、「手続き型」という言葉があります。手続き型ではプログラムをあらかじめ決められた手続きに従って、頭から順に実行します。KAGのシナリオは、手続き型のプログラムです。

イベント駆動型のプログラミングにはデメリットとメリットがあります。イベント駆動型のプログラムは、たとえばGUIのアプリケーションのように「OKボタンが押された」や「メニュー項目が選択された」など、多種多様なイベントが発生する可能性のあるプログラムの記述に向いています。しかし、とくに「何かを待つ」という動作の記述が煩雑になりやすいというデメリットもあります。

KAGのシナリオ・ファイルのように、機能を次々と順を追って実行するような用途では手続き型が適していますが、KAGのシステムそのもののように、GUIを構築する用途としてはイベント駆動型が適しています。

KAGのシステムは、KAGのシナリオ・ファイルに記述された手続き型のシナリオを実行するシステムであり、それ自体はイベント駆動型で記述されているというわけです。

KAGの初期化の過程は「Initialize.tjs」に書いてあります。「Initialize.tjs」の実行が終わると制御は「吉里吉里」に戻り、「吉里吉里」は次のイベントが発生するまで待ちます。何かイベントが発生すれば、「吉里吉里」はTJSで記述されたイベント応答の関数である「イベント・ハンドラ」を呼び出します。

「Initialize.tjs」の実行が終わると、次のイベントが発生するまで待つ…といっても、実際はすぐに「first.ks」の実行が始まるので不思議に思うかもしれません。実はこれは、「Initialize.tjs」を実行し終えた直後、最初のシナリオの開始を告げるイベントがKAG内部で発生するため、「first.ks」がすぐに実行されているように見えるのです。

■ イベント・ハンドラ

イベントに反応して記述される関数、それがイベント・ハンドラです。ウィンドウやレイヤなど、「吉里吉里」が提供するクラスにはよく「on」から始まるメンバ関数があります。これらがイベント・ハンドラです。通常はこれを継承したクラスでオーバーライドすることでイベント・ハンドラを記述します。どのようなイベント・ハンドラが記述できるのか、「吉里吉里2 リファレンス」の「クラスリファレンス」を参照してみてください。

たとえば、「Menus.tjs」には「MenuItem」クラスを継承した「KAGMenuItem」というクラスが定義されています。このクラスはウィンドウ上部に並んでいるメニュー項目を管理するクラスですが、このメンバ関数に、

```
function onClick()
{
        super.onClick(...);
        click();
}
```

というのがあります。これが「イベント・ハンドラ」です。この「onClick」というのは、メニュー項目がクリックされたときに呼び出されるイベント・ハンドラです。ここでは「click();」として、同じクラスのメンバ関数であるclickを呼んでいます。また、イベント・ハンドラでは通常このように親クラスの同名のメソッドを呼び出します。

＊筆者注
"action" メソッドを呼び出すタイプのイベントの処理の方法もあります。

KAGのSystemフォルダにあるTJSスクリプトを見ても、よくこのonから始まる関数が使われているのが分かると思います。

15-2 吉里吉里/KAGの機能のアクセス

TJSからKAGの機能に直接アクセスすることができます。KAGで記述できる範囲のことはKAGのシナリオで記述し、それで物足りなくなったらTJSで「吉里吉里」やKAG内部の機能に直接アクセスできる、というのは、「吉里吉里/KAG」の大きな特徴です。

■ 「Initialize.tjs」と「KAGWindow」

「Initialize.tjs」はKAGの初期化を行なうプログラムです。ほかの各スクリプトの読み込みや二重起動の防止のほか、「f」や「sf」などのKAG変数アクセス用の変数の定義も行なっています。

その中で変数「kag」にクラス「KAGWindow」から作ったオブジェクトを入れている場所を見付けてみてください。

```
/*
        KAG メインウィンドウの作成
*/

var kag;
kag = new KAGWindow();
```

ここですね。この変数kagは、KAGのウィンドウ(普段私たちがゲームなどの作品と接しているあのウィンドウ)そのものです。

この「KAGWindow」というクラスにはKAGの動作のほとんどが定義されていて、多くのKAGの管理するオブジェクト、たとえば背景や前景レイヤ、メッセージ・レイヤ、BGMや効果音用のオブジェクト——などもこの変数kagからアクセスできることになります。

変数kagに入っているのはひな形となるクラス「KAGWindow」そのものではなくて、クラスから作られたオブジェクトであることに注意してください。

「KAGWindow」には「process」というメンバ関数があります。これはKAGのシナリオの実行を開始するための関数です。「Initialize.tjs」の最後では、

```
/*
        first.ks の実行
*/

kag.process("first.ks");
```

となっていて、メンバ関数「process」を呼んでいます。ここから「first.ks」の実行が始まるのです。

■ウィンドウ

「吉里吉里」は、「Window クラス」でウィンドウの機能を提供しています。「KAG」では、「MainWindow.tjs」で「KAGWindow」というクラスを、また「YesNoDialog.tjs」で「YesNoDialogWindow」というクラスを定義しています。

メイン・ウィンドウは、前述の「Initialize.tjs」でKAGWindow クラスから作られています。また、バージョン情報ダイアログも同じKAGWindow クラスのオブジェクトです。

■レイヤ

「Layer クラス」でレイヤの機能を提供しています。「レイヤ」はウィンドウ内に表示されるもので、何かウィンドウ内に表示しようと思ったら、必ずレイヤを作る必要があります。

●背景レイヤ

「背景レイヤ」は、以下の式でアクセスできます。

kag.fore.base	表画面の背景レイヤ
kag.back.base	裏画面の背景レイヤ

背景レイヤは「GraphicLayer.tjs」で定義されている「BaseLayer」クラスのオブジェクトです。

「吉里吉里」のレイヤ管理はレイヤを親子関係で階層的に管理します。その階層のトップ、すべてのレイヤの親にあるのが、「プライマリ・レイヤ」と呼ばれるものです。KAG の場合、「表背景レイヤ」が必ずプライマリ・レイヤになります。

レイヤの階層は、KAGのウィンドウをアクティブにして「Shift + F12」を押すことでコンソールに一覧表示されるので見てみてください。

レイヤは「Layer クラス」から継承されています。たとえば Layer クラスには、レイヤを色で塗りつぶす「fillRect」というメソッドがあります。

これを使って表背景レイヤを真っ赤に塗りつぶすには、

```
kag.fore.base.fillRect(0, 0, 640, 480, 0xff0000);
```

とします。各引数の説明は「吉里吉里２リファレンス」の「Layer クラス」の該当メソッドを参照してください。

● 前景レイヤ

「前景レイヤ」は、以下の式でアクセスできます。

kag.fore.layers[n]	表画面の前景n番のレイヤ
kag.back.layers[n]	裏画面の前景n番のレイヤ

たとえば、「kag.fore.layers[0]」ならば「表画面の０番の前景レイヤ」という意味になります。

「前景レイヤ」は「GraphicLayer.tjs」で定義されている「CharacterLayer」クラスのオブジェクトです。この「CharacterLayer」クラスも Layer クラスから継承されているため、Layer クラスの各メソッドが使えます。

● メッセージ・レイヤ

「メッセージ・レイヤ」は、以下の式でアクセスできます。

kag.fore.messages[n]	表画面のn番のメッセージ・レイヤ
kag.back.messages[n]	裏画面のn番のメッセージ・レイヤ

たとえば、[kag.back.messages[0]] ならば「裏画面の０番のメッセージ・レイヤ」という意味になります。

メッセージ・レイヤは「MessageLayer.tjs」で定義されている「MessageLayer」クラスのオブジェクトです。

■ BGMと効果音

「吉里吉里」は「Wave(PCM)」「MIDI」「CD-DA」のいずれかでサウンドを再生することができます。KAGの場合はBGMにこれらの３つのうちのどれか一つを選べます。効果音にはWave(PCM)のみ使ことができます。

「BGM」や「効果音」には、以下の式でアクセスできます。

kag.bgm	BGM
kag.se[n]	n番の効果音バッファ

BGMを管理するクラス「BGM」は、「BGM.tjs」で定義されています。たとえば“test.wav”をBGMとして再生させたい場合は、

```
kag.bgm.play(%[ storage : "test.wav" ]);
```

で再生できます。

「効果音」を管理するクラス「SESoundBuffer」は「SE.tjs」で定義されています。たとえば“test.wav”を効果音として効果音バッファ0番で再生したい場合は、

```
kag.se[0].play(%[ storage : "test.wav" ]);
```

とします。

■ タグ・ハンドラ

TJSからKAGのタグの機能を呼び出したい場合はよくあります。

KAGのタグの多くは「タグ・ハンドラ」という、各タグに応じた関数を呼び出すことでその機能を実現しています。

タグ・ハンドラは「MainWindow.tjs」の最後のほうにあり、「getHandlers」という関数内に、

```
タグ名 : function(elm)
{
    (ここに処理の内容)
} incontextof this
```

という形式で書き連ねてあります。引数「elm」では、タグの属性が辞書配列になったものを受け取ります。

「getHandlers」はこれらタグ名とそれに対応する関数の辞書配列を返すだけの関数で、この関数の戻り値は変数「tagHandlers」に入れられます。つまり、「kag.tagHandlers」から関数を直接実行してやればよいのです。

たとえば、

```
@image storage=hoge.jpg layer=base page=fore
```

というタグと同様の機能をTJSから呼び出したい場合は、

```
kag.tagHandlers.image(%[ storage : "hoge.jpg", layer :
"base", page :"fore" ]);
```

と記述します。「%[ ]」の中に、

```
属性名 ： 属性の値
```

という形式で属性をカンマで区切って指定して渡すことで、属性名を名前とし、属性の値をそれに対応する値とした辞書配列を渡すことができます。

＊筆者注
%[]内に辞書配列のメンバとなるものを記述して辞書配列を作成する方法を使います。

TJSからタグ・ハンドラの機能を呼び出す場合、「何かを待つ」という種類のもの、たとえばトランジションの終了を待つ「wt タグ」や時間待ちをする「wait タグ」などは正常に呼び出すことができないので、注意が必要です。これらを待つには、これらの完了を知らせるイベントを何らかの方法で受け取る必要がありますが、難しくなります。

■ KAGをTJSから操作するときの注意点

KAGは「栞(しおり)」に多くの情報を保存しますが、KAGが栞に保存しないような情報をTJS側から操作すると、その情報は栞に保存されません。

たとえば、背景レイヤや前景レイヤでは「レイヤにどの画像が読み込まれているか」の情報しか栞に保存されません。なので、たとえばレイヤに直接何かを描画しても、その情報が栞に保存されないため、栞を読み込んだときにそのときの情報が復元されないことになります。この場合は、次の「栞を保存可能なラベル」を通過する前にそのレイヤに別の画像を読み込むなどして、KAGが栞に保存できる情報のみの状態に戻しておく必要があります。

第16章 KAG用プラグイン

KAGの機能を簡単に拡張できるようにするために、KAGはプラグインを扱う機構をもっています。

16-1 KAGのプラグイン

KAGは、KAG用プラグインをTJSで書くことができる機構をもっています。これにより、KAGに簡単に機能を追加できるようになります。

KAGのプラグインは、「KAGPlugin クラス」からクラスを継承して作ります。KAGには「雪プラグイン」や「スタッフロール・プラグイン」などが付属していますが、それらのソースを見てもKAGPluginから何らかのクラスが派生しているのが分かります。

プラグインは、おおむね以下の構造を持っています。

・KAGPluginから派生したクラス本体
・そのプラグインをKAGのシステムに登録する部分
・プラグインを使用可能にするためのタグ(マクロ)の定義

通常は「first.ks」(最初に実行されるシナリオ・ファイル)の最初でプラグインのファイルを呼び出して使います。

■ KAGPluginのメンバ関数

「KAGPlugin」には、「吉里吉里」本体から呼び出されるいくつかの関数があります。たとえば「onRestore」は、栞をたどるときに呼び出されるメンバ関数で、これをオーバーライドして処理を書けば、栞をたどるときに実行したい処理を書くことができます。

以下に「吉里吉里」本体から呼び出される関数の説明をします。また、説明の中で、KAGに付属している「雪プラグイン」である「snow.ks」の内容についても例として説明します。

● finalize

KAGPluginクラスのfinalizeメンバ関数は、通常のクラスでの動作と同じく、オブジェクトが無効化されるときに呼び出されます。プラグインが後述の「kag.addPlugin」でKAGのシステムに追加されると、KAGの終了時に「finalize」が呼ばれるようになるので、ここに何か終了処理を書くことができます。

「雪プラグイン」では、作成したオブジェクトをinvalidate演算子で無効化しています。

● onStore(f, elm)

onStoreは栞を保存する際に呼ばれます。KAGは「保存可能なラベル」(「|」付きのラベル)で栞をいったん内部に保存するので、保存可能なラベルを通過するごとにこの関数が呼ばれることになります。

引数「f」は、保存先の栞データを表わす辞書配列です。KAGの栞が辞書配列なのは前に話したと思いますが、この「f」も辞書配列です。ここに何か辞書配列のメンバを作って情報を記録しておけば、「onRestore」で同じ情報を読み出すことができます。
(なお、引数「elm」は、現バージョンでは使われていません)。

たとえば、「雪プラグイン」では以下のような記述になっています。

```
function onStore(f, elm)
{
        // 栞を保存するとき
        var dic = f.snows = %[];
        dic.init = timer !== void;
        dic.foreVisible = foreVisible;
        dic.backVisible = backVisible;
        dic.snowCount = snows.count;
}
```

「f.snows = %[]」の部分で、「f」に「snow」という名前のメンバを作っています。そこに新しい辞書配列オブジェクトを作っています。KAGのシステムは、栞のデータに辞書配列や配列が入っていると、その辞書配列や配列の中身も保存するので、このような書き方ができます。

変数「dic」にもその新しい辞書配列オブジェクトを入れています。「=」演算子はこのように続けて書くと、いちばん右側に書いたもの(この場合は%[])をそれぞれに代入する、という動作をします。

「dic」や「f.snow」の中身ですが、前に「参照」について話したとおり、

このようにすると「f.snows」と「dic」は同じ辞書配列オブジェクトを参照するようになります。

関数の残りの部分では、その「dic」にいろいろと情報を記録しています。

● onStableStateChanged(stable)

「onStableStateChanged」は、「安定」あるいは「走行中」の状態が変わったときに呼ばれます。引数「stable」は「安定」のときに“真”、「走行中」のときに“偽”になります。

KAG は状態として大きく分けて、「安定」と「走行中」の2つの状態をもっています。

「安定」とは、「s」あるいは「l」あるいは「p」タグで停止中の状態で、ユーザーは栞の保存やメッセージ履歴の閲覧などをすることができます。一方、「走行中」とは、それ以外の状態で、シナリオが実行中であることを示しています。

これらの状態が切り替わるときにこのメンバ関数が呼び出されます。

何か走行中ではやりたくないことがあるときにこの関数を書けば、その状態を知ることができます。

なお、「雪プラグイン」ではこの関数を使っていません (記述は空です)。

● onMessageHiddenStateChanged(hidden)

「onMessageHiddenStateChanged」は、メッセージ・レイヤが「メッセージを隠す」のメニュー項目などで隠されるときと、その状態から抜けるときに呼びされます。引数 hidden は、メッセージ・レイヤが隠されるときに“真”、再び現れるときに“偽”になります。

これは、たとえばプラグインでレイヤを表示していて、それをメッセージと一緒に隠したいときなどに利用することができます。

● onCopyLayer(toback)

「onCopyLayer」は、「backlay」タグあるいは「forelay」タグが実行されたとき、あるいはトランジション終了時に、裏画面の情報を表画面にコピーする必要があるときに呼ばれます。引数「toback」は、「表→裏」のコピーのときに“真”、「裏→表」のコピーのときに“偽”になります。

KAGのトランジションの動作はかなりややこしいので後述します。

「雪プラグイン」では、以下のような記述になっています。

```
function onCopyLayer(toback)
{
        // レイヤの表←→裏情報のコピー
        // このプラグインではコピーすべき情報は表示・非表示の情報だけ
        if(toback)
        {
                // 表→裏
                backVisible = foreVisible;
        }
        else
        {
                // 裏→表
                foreVisible = backVisible;
        }
        resetVisibleState();
}
```

メンバ関数「foreVisible」と「backVisible」はそれぞれ、「表画面の雪粒」あるいは「裏画面の雪粒」が表示状態であるかどうかを表わしています。雪プラグインは雪粒を表示するためのレイヤを、表背景レイヤと裏背景レイヤそれぞれを親にして、表画面と裏画面両方に作りますが、それぞれの可視・不可視の情報をもっているのがこれらのメンバ変数です。

「toback」が“真”の場合は「backVisible = foreVisible」、“偽”の場合は「foreVisible = backVisible」を実行します。「resetVisibleState」メンバ関数は、「foreVisible」および「backVisible」変数の内容を実際に各雪粒のレイヤの表示状態に反映させるメンバ関数です。

● onExchangeForeBack()

「onExchangeForeBack」は、トランジションの終了によって、裏画面と面画面の情報を入れ替える必要があるときに呼ばれます。

トランジションの動作については後述しますが、このメソッドが呼ばれる時点でレイヤのツリー構造は、表画面と裏画面が入れ替わっています。

「雪プラグイン」では以下のように記述されています。

```
function onExchangeForeBack()
{
	// 裏と表の管理情報を交換
	var snowcount = snows.count;
	for(var i = 0; i < snowcount; i++)
		snows[i].exchangeForeBack(); // move メソッド
		                                を呼び出す
}
```

ここでは、各雪粒を管理するオブジェクト(SnowGrainクラスのオブジェクト)の「exchangeForeBack」メンバ関数を呼んでいます。SnowGrainクラスの「exchangeForeBack」は以下のような記述になっています。

```
function exchangeForeBack()
{
	// 表と裏の管理情報を交換する
	var tmp = fore;
	fore = back;
	back = tmp;
}
```

「fore」は表画面に表示されている雪粒のレイヤ、「back」は裏画面に表示されている雪粒のレイヤです。それを交換しています。

トランジションの終了時には裏画面と面画面がそっくり入れ替わっているのですから、このようにレイヤの情報も交換しないと、たとえば、いままで表画面に表示されていたと思っていた雪粒が実は裏画面に表示されていることになっていた、ということになってしまいます。

● onSaveSystemVariables()

「onSaveSystemVariables」は、システム変数に情報を確実に保存するためのタイミングを提供します。システム変数は、KAGが終了し、再び起動しても同じ情報を保っている変数で、何かシステムに関わるような(栞とは独立した)情報を記録しておくのに便利です。

この関数内で「kag.scflags」に何かメンバを作り、そこに情報を記録してお

くことができます。「kag.scflags」は辞書配列オブジェクトです。
「雪プラグイン」にはこの関数はありません。

■ プラグインの登録

プラグイン・クラスを作ったら、そのオブジェクトを作ってKAGのシステムに登録しなくてはなりません。登録するには、「kag.addPlugin」メンバ関数を使います。

「雪プラグイン」では以下のようになっています。

```
kag.addPlugin(global.snow_object = new SnowPlugin(kag));
        // プラグインオブジェクトを作成し、登録する
```

「**global**.snow_object = **new** SnowPlugin(kag)」としてSnowPluginクラスのオブジェクトを作り、グローバル・オブジェクトのメンバ「snow_object」に入れています(このように書くと、「**global**」に「snow_object」というメンバが自動的に作られてからオブジェクトへの参照が代入されます)。

「**global**.snow_object = **new** SnowPlugin(kag)」という部分式が「kag.addPlugin」への引数になっていますが、「=」演算子それ自体は左辺(この場合は「**new** SnowPlugin(kag)」)を返すのでこの引数には新しく作ったオブジェクトが渡ることになります。

この一行を複数行で書くと、以下のようになります。グローバル位置で宣言した変数はグローバル・オブジェクトのメンバになることに注意してください。

```
var snow_object = new SnowPlugin(kag);
kag.addPlugin(snow_object);
```

プラグイン・オブジェクトは二度以上登録してはなりません。二度以上登録しないために、「雪プラグイン」ではTJSスクリプトを定義するiscriptタグを、以下のようなifタグで囲っています。

```
ypeof(global.snow_object) == 'undefined'
```

「**typeof(global**.snow_object)」は、グローバル・オブジェクトのsnow_objectメンバが表わしている型を表わす式ですが、グローバル・オブジェクトに「snow_object」が無ければ"undefined"になるので、それを利用して、すでに

グローバル・オブジェクトにsnow_objectメンバがあるかどうかを確認しています。
iscriptタグ内では最後のほうでsnow_objectメンバを作って値を入れているため、何度実行してもこのiscriptタグは1回限りの実行となります。

■ マクロの登録

プラグインをKAGから便利に使うため、「雪プラグイン」ではマクロをいくつか定義しています。定義は以下のようになっています。

```
; マクロ登録
@macro name="snowinit"
@eval exp="snow_object.init(17, mp)"
@endmacro
@macro name="snowuninit"
@eval exp="snow_object.uninit()"
@endmacro
@macro name="snowopt"
@eval exp="snow_object.setOptions(mp)"
@endmacro
```

マクロの中にevalタグが入っているのに注目してください。evalタグはexp属性で指定されたTJS式を実行するタグです。
たとえばsnowoptマクロでは以下のTJS式を実行しています。

```
snow_object.setOptions(mp)
```

ここで「mp」はマクロに渡された属性が入った辞書配列を表します。
たとえばsnowoptタグを以下のように使うと、

```
alse forevisible=true
```

「mp」が参照している辞書配列のメンバ「backvisible」は“**false**”、「forevisible」は“**true**”になります。
「snow_object」はSnowPluginクラスから作られたオブジェクトで、「setOptions」メンバ関数にこの辞書配列を渡していることになります。

このように、プラグイン・オブジェクトへは簡単にマクロの属性を渡すことができます。

16-2 吉里吉里/KAG のトランジションの動作

プラグインでなにかレイヤを作って使うときに、KAGのトランジションの動作について知っておかなければなりません。

「吉里吉里/KAG」のトランジションの動作はかなりややこしいです。ここでは、trans タグで、「layer=base children=**true**」、つまり背景レイヤとその子レイヤも含めたレイヤに対してトランジションを行なう場合について説明します。

まず、KAGのレイヤの構造ですが、表画面は「表背景レイヤ」を、裏画面は「裏背景レイヤ」を頂点とした「親子関係」(階層構造)となっています。

トランジションの動作は、表画面に表示されている内容が、裏画面に表示されている内容に、時間をかけて切り替わる、ということですね。最終的に、表画面と裏画面の内容は両方とも、「トランジション前に裏画面だったもの」の内容になります。

トランジション中で、切り替わっている間に、レイヤの構造はどうなっているのかというと、表示の状態は変化しているとはいえ、実は構造には変わりはありません。いまだ表画面は表画面、裏画面は裏画面のままです。

構造に変化が起こるのは、トランジションが終了する瞬間です。トランジションが終了した時点では画面はまるっきり裏画面の状態になっているのですから、レイヤの構造的にも裏画面にしなければなりません。

「吉里吉里/KAG」はこのとき、以下の動作をします。

---

①「吉里吉里」は、表背景レイヤとその子レイヤたち、それと裏背景レイヤと子レイヤたちを、そっくりそのまま、入れ替える
②「吉里吉里」は、トランジション前に表背景レイヤだったレイヤの、onTransitionCompleted イベントのイベント・ハンドラを呼ぶ
③KAG は、「onTransitionCompleted」イベント・ハンドラの中で、裏画面だったレイヤの情報を、表画面だったレイヤの情報にコピーする
　(これにより表画面と裏画面が同じ状態になる)
④KAG は、裏画面と面画面の管理情報を入れ替える

最初にも「表背景レイヤとその子レイヤたち、それと裏背景レイヤと子レイヤたちを、そっくりそのまま、入れ替える」とありますが、最後にも「裏画面

と面画面の管理情報を入れ替える」とありますね。これは、①のタイミングで「吉里吉里」本体は裏画面と面画面を入れ替えますが、それだけだと「本体は入れ替えたと思っているが、KAG側は入れ替えたと思ってない」という妙な状態になってしまいます。

いままで裏画面のレイヤ・オブジェクトを参照していたKAG内の変数が、吉里吉里側がレイヤの構造を入れ替えたため、実は表画面のレイヤ・オブジェクトを参照するようになってしまった、という状態を避けるために、つまり、「吉里吉里」本体のもっているレイヤ構造の状態と合わせるために、④のタイミングでKAG側でも処理が必要なのです。

プラグインでレイヤを表示管理する場合もこのルールに従う必要があります。プラグイン側にはこのために2つの関数、「onCopyLayer」と「onExchangeForeBack」が提供されています。「onCopyLayer」は3番のタイミングで、「onExchangeForeBack」は4番のタイミングで呼ばれる関数です。

「onCopyLayer」はforelayタグやbacklayタグでも呼ばれる関数ですが、トランジションの終了時にも呼ばれる関数です。トランジションの終了時においてこの関数では、裏画面だったレイヤの情報を表画面だったレイヤの方にコピーしなければなりません。これは「onExchangeForeBack」よりも先に呼ばれるので、注意が必要です。

つまり、「onExchangeForeBack」で表画面と裏画面の管理情報を入れ替えるのですが、それよりも前に呼ばれるということは、「吉里吉里本体のもっているレイヤ構造の情報と、KAGあるいはプラグインのもっているレイヤ構造の状態が食い違っている状態」で呼ばれるということです。混乱を招くかもしれませんが、通常は、「雪プラグイン」で行なっているように、普通に状態のコピーを行なえば問題ありません。

「onExchangeForeBack」では表画面と裏画面のレイヤの管理情報を入れ替えなければなりません。これは、今まで裏画面のレイヤを参照していた変数を表画面のものに、表画面のレイヤを参照していた変数を裏画面のものにそれぞれ入れ替えなければならないと言うことです。

このように「吉里吉里/KAG」のトランジションの動作はかなりややこしいものですが、「雪プラグイン」などを参考にしてみてください。

16-3 雪プラグインを改造する

プラグインを作るといっても、すべてを何もない状態から書き上げるのは大変です。最初はプラグインの改造から入るのが適切ではないかと思います。

ここではその最初の手引きとして、「雪プラグイン」を改造して「泡プラグイン」を作ってみようと思います。

雪は上から降るものですが、泡は下からわき上がってくるものなので、スクリプトの手直しがいくつか必要になります。

■ 名前の変更

「泡プラグイン」を作るにあたって、いちばん動作が似ている「雪プラグイン」を改造するわけですが、「雪プラグイン」と同時に使いたい場合のことを考えて、名前の衝突が起こらないようにしなければなりません。

まず、「雪プラグイン」のシナリオ・ファイルである「snow.ks」をコピーします。「泡プラグイン」なのでファイル名を「bubble.ks」にしましょう。

bubble.ks 内にはいくつも 「snow」や「雪」などの名前が使われていますが、これを「bubble」や「泡」に変換してしまいましょう。お使いのエディタの「置き換え」機能で以下の名前を一括して変換します。TJSはアルファベットの大文字と小文字を厳しく区別することに注意してください。「置き換え」を行なうときは、大文字と小文字を区別するような動作をさせる必要があります。

変更前	変更後
雪	泡
snow	bubble
Snow	Bubble

このままだと「泡をふらせるプラグイン」のような妙な表現がコメント内に出てきてしまいます。気になる方は書き換えてください。

■ 下から出現するように

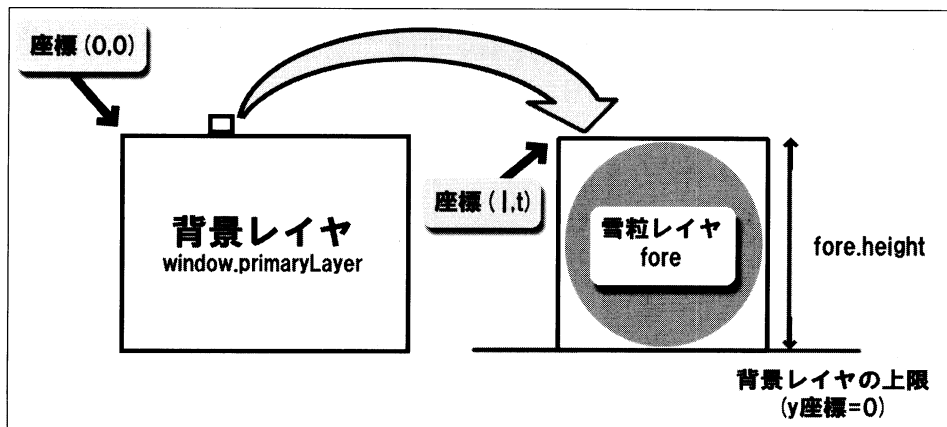
下から出現させるための変更が必要です。

まず、泡を出現させている場所を見てみましょう。BubbleGrain クラスのメンバ関数である「spawn」です。

```
function spawn()
{
        // 出現
        l = Math.random() * window.primaryLayer.width;
                                                  // 横初期位置
        t = -fore.height; // 縦初期位置
        spawned = true;
        fore.setPos(l, t);
        back.setPos(l, t); // 裏画面の位置も同じに
        fore.visible = owner.foreVisible;
        back.visible = owner.backVisible;
}
```

＊筆者注
「吉里吉里」の座標系は、数学等で用いられている座標系と異なり、通常左上を原点とした、右方向および下方向が正方向の座標系です。

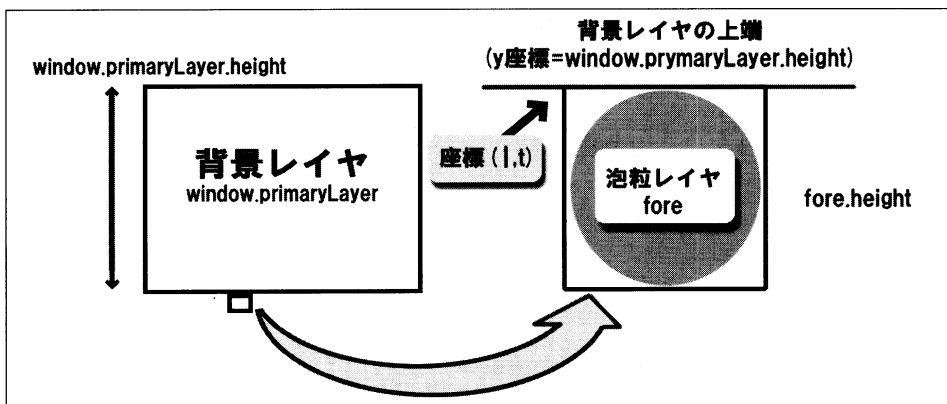
＊筆者注
実際には表画面の粒も裏画面の粒も同じ大きさなので「-fore.height」は「-back.height」でも問題ありません。

ここの「// 縦初期位置」というコメントの付いている行に注目してください。変数 fore は表画面の泡粒のレイヤ・オブジェクトを参照しています。同様に変数 back は裏画面のものです。変数 t は粒のレイヤの y 座標(垂直方向座標)における上端位置を表わす変数ですが、「雪プラグイン」の場合は「-fore.height」、つまり表画面の雪粒のレイヤ・オブジェクトの高さを負にしたものを代入しています。これにより、初期位置は完全に画面外でありながら、画面のすぐ上となっていたのです。つまり、画面外からすぐに画面内に入ってこられる位置にしているということですね (横位置はランダムになります)。

雪粒の出現位置

泡の場合は下からわき上がるのですから、ここを変更しなければなりません。画面外であり、すぐに画面内に入ってこられる位置というと、画面のすぐ下になります。

上端位置が画面の縦幅と同じであればそのような状態になるはずですから、ここを、

```
	t = window.primaryLayer.height; // 縦初期位置
```

にします。「window.primaryLayer」はウィンドウのプライマリ・レイヤ、つまりKAGの場合は背景レイヤですので、その「height」(縦幅) プロパティはすなわち画面の縦幅になります。

泡粒の出現位置

■ 上に向かうように

移動方向を上方向にしなければなりません。

BubbleGrain クラスのコンストラクタ (BubbleGrain メンバ関数) で「yvelo」という、y 方向(垂直方向) の速度を決定している部分があります。以下の部分です。

```
yvelo = n*0.6 + 1.9 + Math.random() * 0.2; // 縦方向速度
```

これの符号をまるっきり逆にしましょう。以下のようになります。

```
yvelo = -(n*0.6 + 1.9 + Math.random() * 0.2); // 縦方向速度
```

■ 画面外に出た場合の処理

雪粒の場合は、雪粒が画面の下を超えれば、雪粒を上の位置に戻して再び降らせていました。その動作を行なっている箇所はBubbleGrain クラスのメンバ関数である move 内にあります。

```
if(t >= window.primaryLayer.height)
{
        t = -fore.height;
        l = Math.random() * window.primaryLayer.width;
}
```

ここの部分です。このif文の条件式は、「雪プラグイン」では以下のようになっています。

```
t >= window.primaryLayer.height
```

変数「t」、すなわち粒のレイヤの上端位置を表わす変数の値が、「window.primaryLayer.height」つまり画面の縦幅を超えたら、という意味になります。

泡の場合は、画面の上端を超えたら、という意味にしなければならないので、条件式は以下のようにしなければなりません。

```
t <= -fore.height
```

すぐに思いつくのは「t < 0」だと思いますが、上端位置が「0」、つまり画面最上部よりも上になっただけではレイヤ全体が画面外に出たことにならないので、「t <= -fore.height」となります。

if文のブロックの中で変数「t」に値を代入していますが、これは前述した関数「spawn」内と同じにする必要があります。

このif文は、最終的には以下のようになります。

```
if(t <= -fore.height)
{
        t = window.primaryLayer.height;
        l = Math.random() * window.primaryLayer.width;
}
```

■ 泡粒の画像

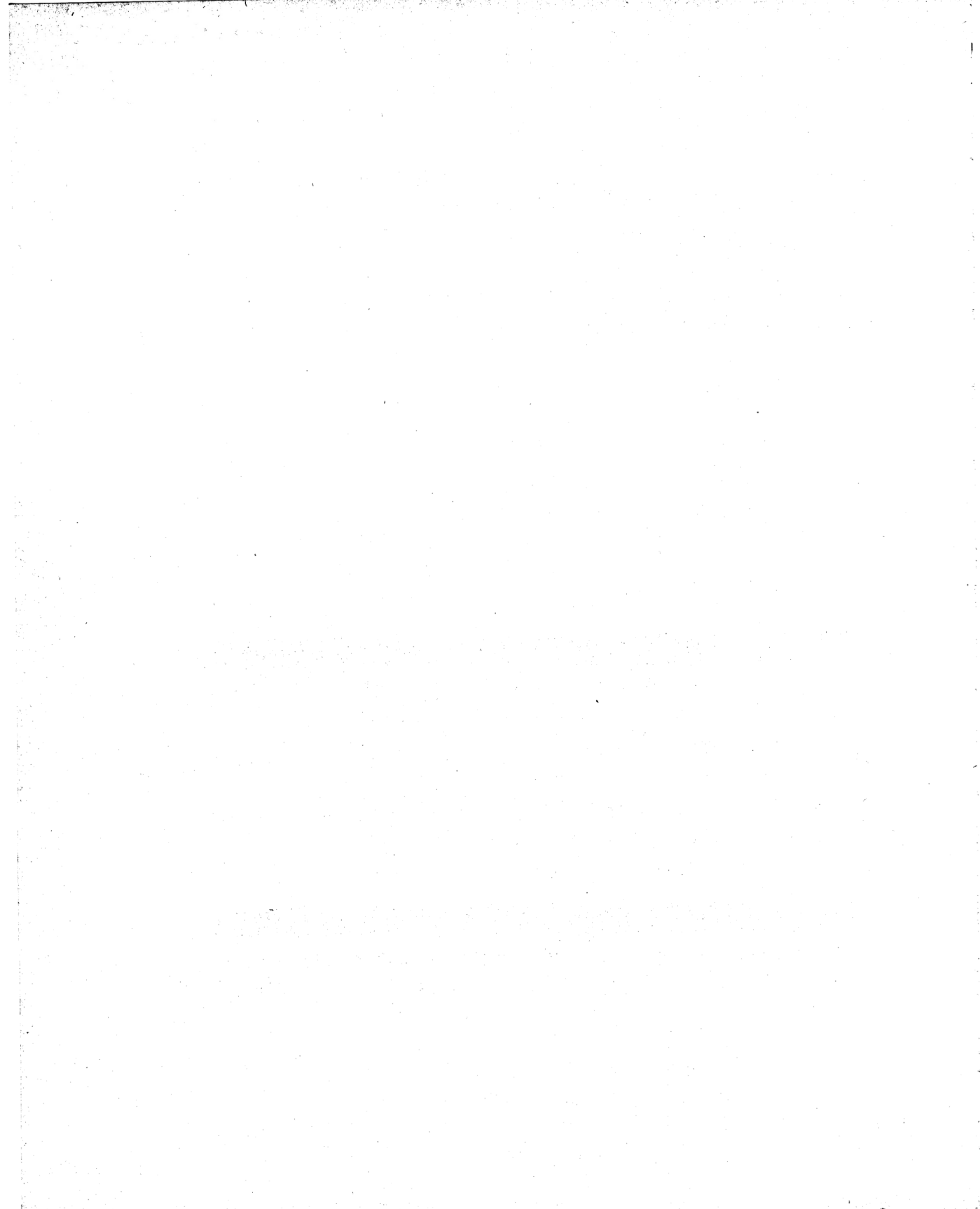
「雪プラグイン」で使う「雪粒」の画像は以下の５つです。

```
「snow_0.png」「snow_1.png」「snow_2.png」「snow_3.png」
「snow_4.png」
```

これと同じように、泡の画像として以下の５つを作ります。

```
「bubble_0.png」「bubble_1.png」「bubble_2.png」
「bubble_3.png」「bubble_4.png」「bubble_5.png」
```

■ 完成

これで改造は終わりです。マクロは「bubbleinit」と「bubbleuninit」と「bubbleopt」の３つが定義されますが、使い方は「雪プラグイン」と同じです。

付録
●FAQ
●TIPS
●CD-ROMの内容について
●収録作品紹介
●演算子一覧
●索引

FAQ

メッセージ・レイヤー/文字関連

Q メッセージ・レイヤーに文字を数段に渡って表示したら、縦列がきれいに揃わない。

A デフォルトにプロポーショナル・フォントを使うと勝手に字詰めが行なわれ、結果として縦列が崩れます。Config.tjs 内のデフォルト・フォント指定を「プロポーショナル」から「等幅」フォントに変更してください。

Q 一般的なアドベンチャー・ゲームのメッセージ・レイヤーを、ゲームの途中でサウンドノベルのように画面全体に広げ、その後、また元の大きさに戻したい。

A メッセージ・レイヤーのサイズをシナリオ内で恣意的に変えるには[position]タグを用います。

●AVG風表示の例

```
[position left=40 top=400 width=540 height=60]¥
```

●ノベル風表示の例

```
[position left=40 top=40 width=540 height=400]¥
```

●全画面表示

```
[position left=0 top=0 width=640 height=480]¥
```

Q メッセージ・レイヤーに文字を書いたあと、トランジションで背景画像を切り替えたところ、すでに書いた文字が消えてしまう。背景だけ入れ替えて、文字はそのまま続けて書きたい。

A トランジションを行なう前に [backlay] タグを入れれば、すでに書いた文字が裏画面にコピーされ、それをトランジションで表画面に呼び出すことができます。ちなみに、[backlay]タグを効果的に利用するには、レイヤーとページの概念を充分に理解することが必要になります。

Q 「吉里吉里2/KAG3」の「KAG2互換モード」で、文末に [p] を付けたのに改ページしてくれない。

A [p] タグは改ページの「クリック待ち」を行なうだけで、「改ページ」までは行ないません。[p] タグの次に [cm] [er] [ct] のいずれかのメッセージ・レイヤー消去タグを置けば、すでに書かれた文字がクリアされます。

Q メッセージ・レイヤーに文字を表示させたところ、文末に改ページマークが出てしまう。互換モードではタグ行末に¥を付けると改ページマークが出なくなったが、KAG3モードでは [position] タグを使う以外に何か方法はないだろうか？

A [style]タグの autoreturn 属性を「false」にすれば改行されなくなります。

Q 選択肢のあるシナリオを書いてから画面をトランジションさせたとき、元の画面の選択肢が見えないのに選べてしまう。

A [locklink] タグを使って選択肢を「選択不可」にすれば回避できます。

Q 前景レイヤーやメッセージ・レイヤーなどに [layopt] タグを使おうとしたら「バリアント型を変換できません」というエラーが出る。

A このエラーは[image] タグにおいて「layer」や「page」属性が指定されていない場合や、属性が指定されていても書き方が間違っている場合に表示されます。また、操作するレイヤーが存在していないことも考えられます。この場合、[laycount] タグで指定のレイヤーを生成すれば回避できます。

Q 自分で作った作品の画面が妙に暗い。

A メッセージ・レイヤーがデフォルトの状態になっていることが考えられます。config.tjs 内のメッセージ・レイヤーの色と不透明度を変更するか、[position] タグを使ってください。

```
// ◆ メッセージレイヤの色と不透明度
// frameColor には 0xRRGGBB 形式で ( RR GG BB はそれぞれ2桁の
// 16 進数)メッセージレイヤの色を指定します。frameOpacity には 0
// ～ 255 の数値で、不透明度を指定します。メッセージ枠用の画像が指定
// されている場合は無効です。
;frameColor = 0x000000; // position タグの color 属性に相当
;frameOpacity = 128; // position タグの opacity 属性に相当
```

↑ここの frameOpacity を 0 にする。

↓または、下記の一行をスクリプトの先頭に加える。

```
[position layer=message0 page=back frame="" opacity=0]¥
```

Q Config.tjs 内で設定した基本フォントを、途中で「Conrfig.tjs」を書き直して変更したのに、変更が反映されない。

A 一度でも「吉里吉里/ＫＡＧ」を動かした場合、「Confif.tjs」内で設定されたフォントはsavedataフォルダ内のセーブ・ファイルにシステム変数として記録されているからです。savedataフォルダ内の全ファイルを削除して再度動作させれば反映されます。

Q 文字入力画面を、ダイアログではなく、インターネットの掲示板に見られるような枠つきの入力ボックスにしたい。

A [edit]タグによる単一行エディットを使います。

Q 文字を縦書きにしたい。

A systemフォルダにある「Config.tjs」の、以下の部分を変更します。

```
// ◆ 縦書きモード
// メッセージレイヤを標準で縦書きモードにする場合は false ではなく
// true を指定してください。
;vertical = true;  // position タグの vertical 属性に相当
```

またはシナリオ・ファイル内に以下のように記述します。

```
[position vertical=true]¥
```

Q メッセージ・レイヤーのサイズを変更した後、初期値にリセットしたい。

A メッセージ・レイヤーの属性を「position」で変更した後、初期設定（Config.tjsで設定したもの）に戻したい時は、再度[position]タグで指定する必要があります。リセットするタグはありません。

Q [if]タグを使って選択肢を表示すると、場合によっては空行が出てしまう。

A [if][link]〜[endlink][r][endif]¥
上のように記述した場合、改行も条件分岐の中に含まれます。条件が“真”の場合は選択肢(〜の部分)が表示されて改行され、条件が“偽”の場合は改行ごと無視されます。

Q メッセージ履歴をクリアしたい。

A 「kag.historyLayer.clear()」で初期化できます。KAGシナリオ内から実行する場合は「[eval exp="kag.historyLayer.clear()"]¥」の一行を記述すれば、その時点でメッセージ履歴が何も書かれていない状態に戻ります。

また栞をたどったときにメッセージ履歴を常にクリアしたいのであれば、「MainWindow.tjs」の「function internalRestoreFlags(f, clear = true, elm = void)」の最後(バージョンによっても若干異なりますが、1283行付近)に「historyLayer.clear();」という一行を加えます。

Q 文字入力ダイアログに入力した文字を、次に入力するときに消したい。

A ダイアログを表示する前に、[eval]タグを用いて以下のように変数に空文字列を入れておけば、何も文字が入力されていない状態でダイアログが表示されます。

```
[eval exp="f.hensuu=''"]
```

Q 終了させたい。

A 2パターンあります。

```
[link exp="kag.close()" ]ゲームを終了する[endlink]
[link exp="kag.shutdown()"]ゲームを終了する[endlink]
```

前者は終了の確認ダイアログが出るのに対して、後者は確認ダイアログが出ません。

Q シナリオを終了するときはどんなタグを打てばよいか。

A シナリオ・ファイルで何も書かれていないところまでくれば、「吉里吉里」はそこで自動的に停止します。[s]タグを記述してもそこでシナリオの実行は停止します。「吉里吉里」自体を終了したければ[close]タグを使い、最初に戻りたいのであれば[jump]タグや[gotostart]タグを使います。

トランジション/アニメーション/画像関連

Q KAG3に付属の雪プラグインを使って画面に雪を降らせ、デバックでシナリオの再読み込みをしたり、開始したラベル以外のラベルからロードすると、雪が止まらなくなる。また、雪プラグインをロードしようとするとエラーが出る。

A 古いKAG3では、雪プラグインを複数回ロードした場合、上記のような現象が起きることがあります。また、KAG3の「ver.3.04」以降では、雪プラグインを2回以上ロードしようとするとエラーが出るようになりました。雪プラグインや雨プラグイン、スタッフロール・プラグインは「first.ks」の先頭などで一度だけ呼び出すようにしてください。

Q mpeg拡張子のビデオ・ファイルを再生しようとしたらエラーが出て再生できない、また、タグが無視されるのだが？

A mpeg形式のビデオファイルを再生する場合はファイルの拡張子まで指定してください。また、「krkr.eXe」と同じ場所に「krmovie.dll」をコピーする必要があります。

Q スクリプトを実行中に「二つのレイヤのサイズは同じでなければなりません」というエラーが出る。

A 背景レイヤーに読み込む画像は画面サイズと同じでなければなりません。また、トランジション前の画像とトランジション後の画像サイズも同じでなければなりません。該当の画像ファイルを画像処理ソフトでリサイズすれば回避できます。

Q クリッカブル・マップの領域画像ファイル（末尾が「_p」のファイル）が上手く作れない。

A PhotoShopやペインターなどのフルカラー対応ソフトではなく、「D-Pixed」（フリーソフト）のような256色専用の画像処理ソフトを使ったほうが簡単です。

Q クロスフェードなどのトランジションを行なったとき、画面にチラツキが発生してしまう。

A 設定ユーティリティであるkrkrconf.exeを起動して、「画像演算の分割処理をする/しない」と「垂直同期を待つ」を調整すれば回避できる場合があります。

Q 画像を表示させようとしたときに「～について適切な拡張子を持ったファイルを見つけられませんでした」のようなエラーが出る。

A 指定のファイルが存在しないか、または見付からないことが考えられるので、ファイル名を確認するか、また、ファイル名を「" "」(半角ダブル・クオーテーション)でくくって指定してください。

Q ルール・ファイルを自作してユニバーサル・トランジションに使ったところ、意図したトランジションが行なわれない。

A ルール・ファイルがグレースケールで保存されていない可能性があります。

Q トランジションを行なったところ、その少し前にトランジションで表示させた画像や文字が改めて表示されてしまう。

A トランジションを行なう前に [backlay] タグを書き忘れている可能性があります。

Q 動画が再生されない。

A 「first.ks」の先頭行に「@loadplugin module=krmovie.dll」を入れてください。このとき、krkr.eXe と同じ場所に、krmovie.dll をコピーする必要があります。この場合、再生する直前に「@eval exp="WaveSoundBuffer.freeDirectSound()"」の 1 行を記述すると、不具合が低減されることがあります。

Q 文字は不透明なままで、自分で作ったメッセージ・ウィンドウを半透明にするには。

A タグで制御するのではなく、メッセージ枠用の画像を半透明の状態で作っておきます。α チャンネル付きの PNG をグラフィック・ソフトに出力させるのが手っ取り早いです。

Q Flash の swf 利用時に、フラッシュ特有の右クリック・ポップアップ・メニューを禁止したい。

A できません。

Q 背景、メッセージ・レイヤー、立ち絵を同時に消したい。

A そのようなタグはありません。baseレイヤーには黒い画像を読み込み、それ以外のレイヤーには不可視状態を指定してトランジションさせれば同時に消えます。

```
[image storage="kuro.jpg" layer=base page=back]¥
[layopt layer=message0 page=back visible=false]¥
[layopt layer=0 page=back visible=false]¥
[trans method=crossfade time=100]¥
[wt]¥
```

Q 640×480の画像が作れない。

A これは画像処理ソフトの使い方の問題ですが、画像処理ソフトで拡大縮小したときにぴったり640×480にならない場合は、リサイズを行なって、余分な部分を切り落とすしかありません。

Q 前景レイヤーの左上に黒い四角が出る。

A 何もデータをロードしていない前景レイヤーにはダミーとして32×32の黒い四角が表示されます。何か画像をロードすれば消えます。

OggVorbis/MP3/音声関連

Q MP3ファイルの再生ができない。

A 吉里吉里2ならびにKAG3において、MP3はβ版の時点でサポートを打ち切っています。OggVorbisという代替手段があるので、そちらをご利用ください。

Q KAGSystemリファレンスの通りに音楽CD（CD-DA）を指定しても、コンソールに「指定されたドライブではCD-DAを再生できません」というエラー・メッセージが出てCDが再生されない。

A CDのドライブが「e:」でないことが考えられます。もしCDドライブに、「f:」や「q:」などの別のドライブレターを与えている場合は、スクリプト内の「e:」をそのドライブレターに書き換えれば動作します。

Q Waveファイルの BGM や効果音が途切れ途切れに再生されたり、再生中につっかえるような現象が起こるのだが？

A 「吉里吉里2」に同梱されている「krkrconf.exe」あるいは「userconf.exe」を起動して、「DierctSound」周りのオプションを変えてみてください。

Q OggVorbis による BGM や効果音が再生されないのだが？

A 指定のプラグインをシナリオ先頭部などでロードしているか確認してください。プラグインがロードされていないと、OggVorbis形式のファイルは利用できません。

起動/動作/バージョンアップ関連

Q 「吉里吉里」の動作が急に重くなったり、マウスが動かなくなったりする。

A ウイルス活動を常時監視するソフトウェアの中には、アプリケーションの動作を極端に遅くするものもあります。これをはじめ、できるだけ多くの常駐ソフトをタスクトレイから外してください。

Q KAG で書いたシナリオを「吉里吉里」で動かしたら、毎回シナリオが first.ks から始まってしまう。

A 「first.ks」内で [jump] タグを使って別のファイルへ飛べばよいかと。

Q 作成したゲームを CD-R に焼いて実行すると、終了しようとしたときと栞を挟もうとした時にエラーが出るのだが？　CD-R の中身をハードディスクにコピーして実行してもエラーが出るのだが？

A CD-R上から実行すると、吉里吉里が生成したシステム・データを書き込めずにエラーが出ます。「Config.tjs」の中身で栞を使わないように指定すれば回避できる場合があります。また、ハードディスクにコピーしただけではCD-R上の読み取り属性が解除されないので、同様にエラーになります。ファイルの読み取り属性を解除するようにしてください。

Q 「吉里吉里」の紹介用プロジェクト「syokai」を起動しようとすると、エラーが出て起動できない。

A 「吉里吉里」の紹介用プロジェクト「syokai」は、KAGのtemplateフォルダにあるシステムや「KAG3plugin」にあるスクリプトなどを使うようになっているため、KAGのフォルダ構成を変更した場合は動作しません。KAG3の圧縮ファイルを展開した直後に見られるフォルダ構造を維持していれば、動作します。

Q 吉里吉里/KAGを起動したとき、「～はXP3アーカイブではないか、対応できない形式です」というエラーが出て起動できないのだが？

A 吉里吉里/KAGでプロジェクトを起動する時に指定するのはプロジェクト・フォルダであって、個々のシナリオ・ファイルではありません。フォルダを指定してください。

Q 新しいバージョンに入れ替えたら動作がおかしい、動作しない。

A アップデートに失敗している可能性があります。KAGのアップデートをする場合は、基本的には、新しいKAGのsystemフォルダの中身を、既存のsystemフォルダに上書きするだけでOKです。また、「krkr.eXe」も忘れずに上書きを。

Q レンダリングずみフォントが割り当てられない。

A first.ksの冒頭などに拡張子まで含めて記述します。

```
[mappfont storage="font.tft"]¥
```

Q 起動時にフルスクリーンにしたい。

A 手っ取り早い方法は、フルスクリーンの状態の時に終了させ、そのときの「datasc.kdt」(「datasc.ksd」の名前かも知れません)をセーブ・データとして一緒に配布します。「datasc.kdt」や「datasc.ksd」には、前回フルスクリーンだったかどうかの情報などが書き込まれています。

Q 画面がウンともスンとも言わなくなってしまう。フリーズしたわけではないのだが。

A SHIFTキーとF4キーを同時に押してコンソールを表示させてください。コンソールの最下部のメッセージを読むと、「非表示になっているメッセージ・レイヤーの裏ページなどでクリック待ちになっている」場合があります。これはトランジションのミスなどでよく発生します。コンソールに書かれたメッセージに従って対処してください。

Q 雪や雨プラグインが利用できない。

A 雪／雨プラグインのシナリオ・ファイルと画像ファイルを自分のプロジェクト・フォルダにコピーし忘れている可能性があります。

Q ロード時に確認ダイアログを表示させたくないのだが？

A
```
[link exp="kag.restoreBookMark(0, false)"][emb exp="kag.getBookMarkPageName(0)"][endlink]
```
と書きます。「restoreBookMark」と「storeBookMark」は第二引数でダイアログを出すかどうか判断しますが､これらはデフォルト設定では「true」になっています。

Q 実行ファイルの容量を減らしたい。

A 画像データの場合、画質を落としてもかまわないならば､「PNG」「TLG」「ERI」ではなくJPEGを使うとサイズを軽減できます。音声の場合は、無圧縮wavを使うよりは「OggVorbis」などを使うとサイズを軽減できる場合があります。また、Releaserで出力する実行（exe形式）ファイルはzipと同じ圧縮方法を使っているため、zip程度の圧縮しかかけることができません。もっと圧縮をかけたい場合は、Releaserの設定で「圧縮」に分類されているファイルはすべて「無圧縮」に分類し、配布するときにcabなど圧縮率の高いアーカイバで圧縮し直すとよいでしょう。UPXよりもcabのほうが圧縮率が高いです。

その他/素朴な疑問

Q シナリオ・ファイルは大きすぎると処理的に辛いと聞くが、一つのシナリオ・ファイルの適切なサイズはどれくらいか。

A 一つのシナリオ・ファイルが200KBを超えるようなら、分けたほうがよいでしょう。ちなみに、処理的に辛いと言ってもそう問題になるほどではないので、あまり気にする必要はありません。最近のバージョンではむしろ他のシナリオ・ファイルにジャンプするときに時間がかかる場合があります。とは言っても、こちらもそう問題になるほどではありません。

Q 「吉里吉里」はノベルゲームやアドベンチャー・ゲームに特化したエンジンなのか。

A 違います。ノベルゲームに特化しているのは､「吉里吉里」ではなくて「KAG」のほうです。「吉里吉里」はアプリケーションやゲームのエンジンで、それをノベルやアドベンチャー・ゲームの専用エンジンとして動作させるのが､「KAG」です。

TIPS

■画像の一部を消去する

何かの画像を読み込んだレイヤーに対し、矩形で指定した任意の部分の画像を消去（透明処理）できますが、TJSでレイヤーに直接描画する必要があります。「face=dfBoth」にして、Layerクラスのメンバ関数である「colorRect」の不透明度の引数に負の数を指定すると、対象の矩形領域から不透明度が取り除かれます。この場合は色の引数は無視されます。

■クリッカブル・マップでカーソルのロール・オーバー時に画像を変える

領域アクション定義ファイルで「onenter」を使うとよいと思います。

```
1: onenter="kag.fore.layers[0].loadImages(%[storage:'map1.png'])";
```

「領域1」にカーソルが来たとき、表画面の「前景レイヤー0」に「map1.png」を読み込んでいます。ちなみに[button]タグで対応できるような内容であれば、buttonタグを使ったほうが楽です。画像ファイルに「map.png」を用意し、それとは別に、変えたい画像ファイルを「map1.png」などで用意します。

●シナリオ例

```
*start
@layopt layer=0 page=fore visible=true
;↑前景レイヤー0を表示
@layopt layer=message0 page=fore visible=false
;↑邪魔なので消去
@image layer=base page=fore visible=true storage=map
@s
*m1
@layopt layer=message0 page=fore visible=true
領域1をクリック
@s
*m2
@layopt layer=message0 page=fore visible=true
領域2をクリック
@s
```

「map.ma」の内容は以下のようになります。

1	onenter="kag.fore.layers[0].loadImages(%[storage:'map1.png'])"; stor age ="first.ks"; target="*m1";
2	nenter="kag.fore.layers[0].loadImages(%[storage:'map2.png'])"; stoage="first.ks"; target="*m2";

■前景立ち絵の“口ぱく”を効率よく行ないたい

「セリフをしゃべりはじめた」ときにアニメーション・ファイルの付属している画像を[image]タグでロードし、「セリフをしゃべり終わった」ときにアニメーションしていない静止画像をロードすれば、効率的です。

■レイヤー上に簡単な図形を書きたい

TJS2を用いれば可能ですが、実用的な速度は出ません。

```
@eval exp="kag.fore.base.colorRect(10, 20, 100, 100,
0xff0000, 128)"
```

上の記述の意味は「表背景レイヤの(10,20) - (109, 119)を赤、不透明度50%で塗りつぶす」になります。吉里吉里本体に矩形と点や変形以外の描画を行なう機能を追加する予定は今のところありません。

■画像の階調変更

階調の反転はimageタグの「rceil」「gceil」「bceil」にそれぞれ「0」を、「rfloor」「gfloor」「bfloor」にそれぞれ「255」を指定することによって可能になります。その他、値を調整すればセピア調への変換などもできます。

■「吉里吉里」の起動画面を黒から白にする

「Initialize.tjs」というファイルの末尾にある「KAG.process("first.ks");」という部分を「KAG.process("first.ks",,,true);」に書き換えてください。

また、Config.tjs内の下記の部分を「;initialMessageLayerVisible = false;」とします。その後、「first.ks」(最初に実行されるシナリオ・ファイル)の先頭行で表背景レイヤーに白い画像を読み込むように記述します。

このとき初期状態ではメッセージ・レイヤーが表示された状態になっているので、「表メッセージ・レイヤー0」も非表示にしたほうがよいでしょう。その後、[wait]で少しウェイトをとって、確実に画面が表示されるようにします。

●シナリオ例

```
[image stORage="white" page=fORe layer=base]¥
[layopt layer=message0 page=fORe visible=false]¥
[wait time=100]¥
```

■[s]タグのみ停止させる仕様

未読既読とは無関係に、単に「次の選択肢まで進む」仕様にできます。つまり、[s]タグでのみ停止する仕様ですが、これは以下のようにすれば実現できます。

①「Config.tjs」の「autoRecordPageShowing」を「false」に設定。
②さらに[wb]タグの属性を「canskip=true」にする。

■買い物画面での買い物制限

買い物などの画面で持ち金が足りなくなるとリンクを無効にし、品物を買えなくするようなシナリオは、下記のように記述します。

●シナリオ例

```
;変数を初期化
[eval exp="f.a=0,f.b=0,f.money=2000"]
;代金を決める
[eval exp="f.ia=80,f.ib=1000"]
*loop
[er]
[nowait]
卵　[emb exp=f.a]個/[emb exp=f.ia]円
[if　exp="f.money　[if　exp="f.money>=f.ia"][link
target=*plusa]購入[endlink][endif][r]
エプロン　[emb exp=f.b]個/[emb exp=f.ib]円
[if　exp="f.money　[if　exp="f.money>=f.ib"][link
target=*plusb]購入[endlink][endif][r]
残金[emb exp=f.money]円
[endnowait]
[s]
*plusa
```

```
[eval exp="f.a=f.a+ 1"]
[eval exp="f.money=f.money-f.ia"]
[jump target=*loop]
*plusb
[eval exp="f.b=f.b+1"]
[eval exp="f.money=f.money-f.ib"]
[jump target=*loop]
```

■入力文字の制限

単一行エディット（EditLayer.tjs）に「ひらがな」と「・」と「ー」のみ入力させ、それ以外の文字が入力されたら弾く場合は、文字コードをチェックして下記のように書くことができます。**272行目**(関数「onKeyPress」)の、

```
if(#key >= 32)
```

となっているのを、以下のように書き換えます。

```
if(key >= '¥x3040' && key <= '¥x309f' || key == '・' ||
key == 'ー')
```

■リンクや選択肢への自動フォーカス

複数の選択肢を表示させ、指定した選択肢に自動的にカーソルをフォーカスすることができます。選択肢を表示した直後に「@eval exp="kag.fore.base.cursorX = 40, kag.fore.base.cursorY = 40, kag.current.keyLink=0"」と書き、選択肢のある場所までカーソルを移動させ、kag.current.keyLink=の部分でリンク番号を指定します。

ちなみに、リンク番号は1番目のリンクが「0」、次からは「1」「2」「3」となります。

■リンクや選択肢用矩形の調節

リンクを選択したときに出る半透明の矩形の長さを調節するには、[link]と[endlink]の間に[locate]タグを使います。

●シナリオ例

```
[style autoreturn=false]
[link]ほげら[locate x=0] [locate x=580] [endlink][r]
[link]ほげらうね[locate x=0] [locate x=580] [endlink][r]
```

```
[link]ほげらうねら[locate x=0] [locate x=580] [endlink][r]
[style autoreturn=true]
[locate]タグのあとの半角スペースをそれぞれ忘れないように。
```

■選択肢のランダム表示

```
あめだまを

A. 綾里姉にあげる
B. 吉里ちゃんにあげる
C. Deeちゃんにあげる
```

というのを、選択肢のアルファベットの部分はそのままで、選択内容だけを毎回ランダムに入れ替えて表示させたい場合、以下のように2次元配列や「入れ替え」演算子を使うとスマートに書けます。

●シナリオ例

```
@eval exp="f.data = [['綾里姉にあげる', 'ジャンプ先ファイル名.ks', '*ジャンプ先ラベルA'], ['吉里ちゃんにあげる', 'ジャンプ先ファイル名.ks', '*ジャンプ先ラベルB'], ['Deeちゃんにあげる', 'ジャンプ先ファイル名.ks', '*ジャンプ先ラベルC']]"

@eval exp="f.tmp = intrANDom(0,2), f.data[0]  f.data[f.tmp]"
@eval exp="f.tmp = intrANDom(0,2), f.data[1]  f.data[f.tmp]"
@eval exp="f.tmp = intrANDom(0,2), f.data[2]  f.data[f.tmp]"

A . [link target="[0][2]" stORage="[0][1]"][emb exp="f.data[0][0]"][endlink]
B . [link target="[1][2]" stORage="[1][1]"][emb exp="f.data[1][0]"][endlink]
C . [link target="[2][2]" stORage="[2][1]"][emb exp="f.data[2][0]"][endlink]
```

1行目で、「選択肢として表示する文字列」、「ジャンプ先のシナリオ・ファイル」、そして「ジャンプ先ラベル」を定義しています。中央の3行で、1行目で定義した内容を乱数を使ってシャッフルしています。単純に個々の乱数から結果を得るわけではないので、重複はありません。最後の3行で、シャッフルされた内容を選択肢として表示します。

■BGMのランダム再生

前述のTIPS「選択肢のランダム表示」を応用し、選択肢を表示する代わりにPLAYBGMを実行するように指定すれば可能です。

●シナリオ例

```
@eval   exp="f.myBGM   =   ['bgm1.mid',   'bgm2.mid',
'bgm3.mid']"
@eval  exp="f.tmp  =  intrandom(0,2),  f.myBGM[0]  <->
f.myBGM[f.tmp]"
@eval  exp="f.tmp  =  intrandom(0,2),  f.myBGM[1]  <->
f.myBGM[f.tmp]"
@eval  exp="f.tmp  =  intrandom(0,2),  f.myBGM[2]  <->
f.myBGM[f.tmp]"
```

■リンクのツール・チップ

「[link hint="&' 1 行目¥n 2 行目'"]ヒントヒント[endlink]」のように、「hint=」に続けて「"&'」と「' '」で囲んでヒントを書くことができます。途中で「¥n」を記述すると、そこで改行させることができます。

■一度入力した名前をそのまま使う

2回目以降のプレイ、または一度名前を入力した場合に、最初に定義されているサンプル文字をプレイヤーが入力した名前に入れ替えることができます。名前を格納する変数を、ゲーム変数"f.name"ではなく、システム変数"sf.name"にすれば可能です。

■ラベル名を乱数から生成する

以下のように記述します。

```
[eval exp="f.ikisaki='*number'+intrandom(1,3)"]
```

イコールの右側では乱数によって得られた1から3の数字と「*number」というラベル名（文字列扱い）を結合し、それぞれ「*number1」「*number2」「*number3」というラベル名を生成しています。

それを「f.ikisaki」という変数に代入しているので、「[jump target="&f.ikisaki"]¥」と書けば、「f.ikisaki」という変数内のラベル名が「&」によって[jump]タグのtarget属性に引き継がれます。

当然、「*number1」「*number2」「*number3」というラベルがシナリオ内に存在しなければなりません。

■ラベル名を変数で指定する

乱数だけでなく、一般の変数も、同じようにラベル名に利用することができます。

●シナリオ例

```
[input name=f.name]¥
[eval exp="f.ikisaki='*'+f.name"]
[jump target="&f.ikisaki"]¥
*minako
[er]¥
みなこ[s]
*emi
[er]¥
えみ[s]
*ai
[er]¥
あい[s]
```

ただし、この例だと、準備されていない文字を入力された場合に飛び先のラベルが存在しないため、エラーになります。

■[move]タグのpath属性に変数を用いる

「path」は、「カンマ」「空白」「(と)」のいずれかで区切られた数値の列を受け付けるので、

```
@eval exp="f.x=400, f.y=400, f.o=255"
@move layer=0 spline=false page=fore delay=0 time=1000
path="&f.x+','+f.y+','+f.o"
```

などのように、カンマで区切った変数を渡せます。

■オリジナルマクロのパラメータを変数に代入する

mpを用います。「[macro name=piyo][eval exp="f.test=mp.para"][endmacro]¥」と定義し、シナリオ内で[piyo para=une]と書けば'une'という値がf.testに入ります。

■0詰め／空白詰め

「[emb exp="'%.2f'.sprintf(f.tmp)"]¥」のように記述します。「%.2f」は小数点以下２桁まで表示するという意味です。同様に、

"%2d"	(2桁に満たない場合は上位を空白詰め)
"%3d"	(3桁に満たない場合は上位を空白詰め)
"%03d"	(3桁に満たない場合は上位を0詰め)

のように書くことができます。

■入力文字列を数値として判定

文字列を数値に変換するには「+」か「str2num」を使います。「f.number = +f.number」あるいは「f.number = str2num(f.number)」のように記述します。「+」は半角しか受け付けませんが、「str2num」は入力された文字が全角でも強制的に数値へ変換します。

■変数内容を含むセーブ・タイトル

「sf.name2」が登場人物の名前だとすると、「*label|&f.name2+'の提案'」のように記述すれば、ラベルのセーブ・タイトルに変数の内容が代入されます。

■変数内容の距離

プレイヤーが入力した変数を「f.nm1」とし、乱数によって得られた変数をf.nf1～f.nf3としたとき、もっとも「f.nm1」に近い変数をf.nf1～f.nf3の中から選ぶには、次のようにシナリオに記述します。

●シナリオ例

```
[eval exp="f.nc11=Math.abs(f.nm1-f.nf1)"]¥
[eval exp="f.nc12=Math.abs(f.nm1-f.nf2)"]¥
[eval exp="f.nc13=Math.abs(f.nm1-f.nf3)"]¥
[eval exp="f.nmin=Math.min(f.nc11,f.nc12)"]¥
[eval exp="f.nmin=Math.min(f.nc13,f.nmin)"]¥
[if exp="f.nmin==f.nc11"]nf1[endif]¥
[if exp="f.nmin==f.nc12"]nf2[endif]¥
[if exp="f.nmin==f.nc13"]nf3[endif]¥
```

■最初に起動された日から経過日数を求める

シナリオの先頭などで「[eval exp="sf.firstTime = (new Date()).getTime() if sf.firstTime === void"]¥」と書いておけば、システム変数の「firstTime」に初回起動時の日付と時刻が入ります。時間原点を「1970年1/1の午前0:00:00」とし、それからの経過ミリ秒を表わしているので、現在との差を取ることで時間差を計算します。「(new Date()).getTime()」の部分で現在の時間原点からの経過ミリ秒を得ています。

```
[eval exp="f.passedTime = (new Date()).getTime() -
sf.firstTime"]¥
[eval exp="f.passedDates = ~~(f.passedTime /
(60*60*24*1000))"]¥
初回起動から [emb exp="f.passedDates"] 日¥
```

上記の3行を記述することにより、初回起動時からの経過日数を求めることができます。

■waitした時間を変数に代入する

waitに入る直前に、変数に「System.getTickCount()」で得られる値を代入します。

そのあと、リンクで飛んだ先でもう一度「System.getTickCount()」で値を得、その値から先ほどの値を減算すると二つの地点間の通過時間を得ることができます。

●シナリオ例

```
[link target=*s2]リンク[endlink]
[eval exp="f.start = System.getTickCount()"]¥
[wait time=5000]¥
[s]
*s2
[eval exp="f.elapsed = System.getTickCount() - f.start"]¥
; この時点で f.elapsed に通過時間
[emb exp="f.elapsed"]¥
[s]
```

■wait長を変数で指定したい

エンティティを利用します。

●シナリオ例

```
*a
[eval exp="f.a =1000"][wait time="&f.a"]
こんにちは
[l]
```

また、配列を使う方法もあります。

●シナリオ例

```
*a
[eval exp="f.a =[]"][eval exp="f.a[1] =10"]
[eval exp="f.a[2] =1000"][eval exp="f.a[3] =10000"]
[eval exp="f.b =1"][wait time="&f.a[f.b]"]
こんにちは
[l]
```

こちらの場合は、[eval exp="f.b =1"]の「1」を「2」や「3」にすることによってウエイトを変えることができます。

■「待ち」のときに栞をいじらせない

[waitclick]タグを使います。このタグが効いている場合は、栞の操作などは一切できません。

■通常時とノーウェイト時で処理内容を変更する

「[if exp="kag.chSpeed!=0"]ノーウェイトのとき行なわない処理。[endif]」または「[playse storage="再生するファイル" cond="kag.chSpeed!=0"]」と記述します。

「kag.chSpeed」というのは、文字表示速度です。

■ビデオ再生開始時の黒バックを表示しない

映像の再生開始時に一瞬黒くなる状態を回避するには、再生開始時を見せないようにします。下記のように再生開始後わずかな時間だけ映像を非表示にしておけば黒バックを見えなくできます。

●シナリオ例

```
[video left=0 top=0 width=640 height=480 visible=false]¥
[playvideo storage="white.avi"]¥
[wait time=100]¥
[video left=0 top=0 width=640 height=480 visible=true]¥
```

■曲と画像の同期

[resetwait] タグと [wait mode=until time=?????] を組み合わせて使います。

●シナリオ例

```
[playbgm storage="xxxxxx" loop=false]¥
[resetwait]¥
; なんらかの処理
[wait mode=until time=????]¥
; ↑曲再生開始(正確には resetwait タグの位置)から
; ???? ミリ秒まで待つ
```

■音と文字を同期させ、スキップ中は音声を再生させない

自動的に読み進む場合に“真”になっている「kag.autoMode」を利用します。

●シナリオ例

```
[playse storage=voice.wav cond="!kag.skipMode"]
表示メッセージです！
[ws cond="kag.autoMode"][l][stopse]
```

自動読み進み状態の場合は[ws]タグで待ち、スキップ動作中は音声を再生しないようにcond属性を指定しています。

■動画や音声データをCDから読み込む

必要最小限の実行ファイルのみをハードディスクにインストールし、それ以外の画像や音声データをCDにアクセスして読み込みたい場合は、下記のようにします。

要領は、「Storages.searchCD」で返ってきたドライブ文字を、「Initialize.tjs」の「Storages.addAutoPath」の指定先に組み込むことです。「Storages.searchCD」は、該当するCDが見付からなかった場合に空文字列を返します。

●シナリオ例

```
*checkCD
@eval exp="f.CDdrive = Storages.searchCD('GAME_CD')"
@if exp="f.CDdrive == ''"
@cm
[nowait]CD をドライブに挿入してください...[endnowait][l]
@jump target=*checkCD
@endif
@eval exp="Storages.addAutoPath(f.CDdrive + ':/data/')"
@eval exp="Storages.addAutoPath(f.CDdrive + ':/data/some.
xp3#')"
@s
```

「GAME_CD」はCDのボリューム・ラベルです。「Storages.addAutoPath」を使っている部分でCDのcgディレクトリと「data/some.xp3」の中に自動検索パスを設定しています。自動検索パスを設定しておけば、ファイル名を指定すれば自動的にCDの中のファイルを参照するようになります。

■2つのXP3ファイル間でシナリオ・ファイルのジャンプ

TJSスクリプトを変更すれば可能です。「Initialize.tjs」の「if(Storages.isExistentStorage(System.exePath+"patch.xp3"))」と書いてある行の前に、たとえば以下のように記述します。

```
if(Storages.isExistentStorage(System.exePath        +
"scenario.xp3"))
{
Storages.addAutoPath(System.exePath + "scenario.xp3#");
}
```

これで「吉里吉里」の実行可能ファイルと同じ場所に「scenario.xp3」が存在すれば、その中にあるファイルも読みに行くようになります。

■配布ファイルを分割する

たとえば、「scenario.xp3」と「graphic.xp3」を作っている場合、

```
Storages.addAutoPath(System.exePath + "scenario.xp3#");
Storages.addAutoPath(System.exePath + "graphic.xp3#");
```

のような行を、「Initialize.tjs」に「追加」する必要があります。

場所は「Initialize.tjs」のどこでもよいですが、書く位置によってファイルの検索順が変わるので、同名のファイルが存在するときは注意が必要です。他の「Storages.addAutoPath」よりも後ろに書いたほうが検索で優先されるので、

```
if(Storages.isExistentStorage(System.exePath +
"patch.xp3"))
```

という行の直前に追加することをお勧めします。

必要に応じてこれらは追加してかまいませんが、このような記述だと、各xp3内直下のファイルしか検索されません。xp3ファイルは内部でフォルダの階層構造をもてるようになっているので、「scenario.xp3」や「graphic.xp3」を作るときは、Releaserで指定するとき、その指定したフォルダ直下に必要なファイルが存在するようにしなければなりません。必要ならば、

```
Storages.addAutoPath(System.exePath +
"graphic.xp3#graphics/");
```

のようにアーカイブ内のその下の階層を指定することもできます。

また、「Storages.addAutoPath」には、「System.exePath +」が必要になります。「吉里吉里」の実行可能ファイルと同じ場所にあるということを明示しないと、トラブルが発生する可能性があります。

■複数のパッチを配布する

初回公開時のファイルを「ver1.0」としてver1.1パッチを作り、公開した後に、ver1.2パッチをリリースしたくなった場合には、ver1.2パッチの内容にver1.1の変更点を包有しないでもver1.1から1.2への差分ファイルだけの配布ができます。

KAG3の場合、patch.xp3の次にpatch2.xp3 patch3.xp3 patch4.xp3……と探していきます。なかった場合は、そこで探すのをやめます。このため、パッチだけを随時追加していくことができます。

Warning!(注意)

このページ以降に紹介されている作品は、KAG使いを目指して吉里吉里2/KAG3を学習される皆さんのために、フリーソフト作者の方や同人サークルの人々が「参考までに」と本書への収録を許可してくださったものです。

中にはソース（シナリオ・ファイル、画像、音など）を全部または一部公開している作品もありますが、以下の利用規定に従ってご活用ください。遵守されない場合は著作権の侵害となります。

①本書に付属するCD-ROM内に収録されているゲーム・作品・素材については、それぞれの作者または権利者がreadmeファイルなどのドキュメントに明記している利用規定に従って利用してください。

②特に作者の明記がない場合、これらの一部または全部を不特定多数の人間が閲覧可能なネットワーク上に無許可でアップロードしてはなりません。また、無許可でCD-ROMなどのメディアに収録し再配布してはなりません。

③画像、音響素材、映像などの素材をそのまま、または改変して自作品内に使用してはなりません。個人的な学習用途に利用する場合のみ自由に利用できます。

④シナリオksファイルの内容をコピーしてそのまま自作品内に使用してはなりません。自作品に合わせて改変し、利用する場合は可です。収録されたシナリオksファイルの利用はあくまでも参考程度にとどめてください。

⑤このCD-ROM内のデータを利用したことによってユーザーが何かしらの損害を蒙った場合、本書の著者、本書の出版社、収録ソフトウェアの作者は何らその責を負わないものとします。この条件に同意できなければデータの利用は認めないものとします。

KAG使いを目指して後に続く人々のためにも、ルールを守って気持ちよく利用されることをお願いいたします。

CD-ROMの内容について

▼収録データ・フォルダ

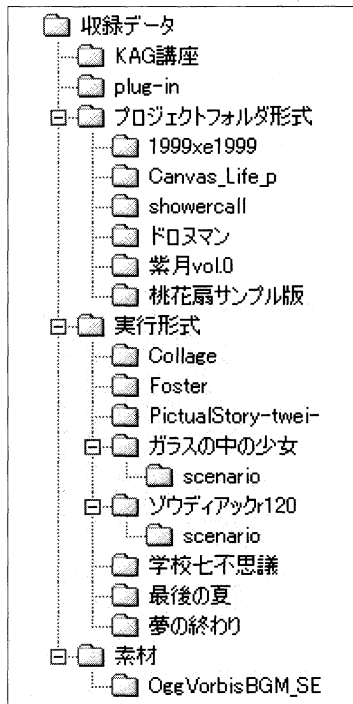
吉里吉里ダウンロードサイトで公式に配布されている「吉里吉里2 SDK version 2.18 / KAG3 version 3.18 revision 2」と、ルール・ファイルが30種類以上含まれたトランジション・ライブラリが収録されています。

利用規定については同梱のドキュメント・ファイルの内容に従ってください。

```
収録データ
├KAG講座
├plug-in
├プロジェクトフォルダ形式
│├1999xe1999
│├Canvas_Life_p
│├showercall
│├ドロヌマン
│├紫月vol.0
│└桃花扇サンプル版
├実行形式
│├Collage
│├Foster
│├PictualStory-twei-
│├ガラスの中の少女
││└scenario
│├ゾウディアックr120
││└scenario
│├学校七不思議
│├最後の夏
│└夢の終わり
└素材
　└OggVorbisBGM_SE
```

▼KAG講座フォルダ

本書のKAG講座で説明されている内容を用いたサンプル・プロジェクトが収録されています。これは本書内に掲載されているシナリオ例の大半をそのまま吉里吉里で動かせるようにしたものなので、本書を読みながらStepごとにひとつずつ実行してみることをお勧めします。

なお、本プロジェクトはいくつかの吉里吉里プラグインを利用しています。このため吉里吉里を展開したときに得られる「kr2_218r2¥kirikiri2¥plugin」内に含まれているすべてのプラグインを「krkr.eXe」と同じフォルダ内に配置しないと正常に動作しません。本プロジェクトの利用規定については同梱のドキュメント・ファイルの内容に従ってください。

▼plug-inフォルダ

TJS編第16章で制作演習を行なった「泡のプラグイン」が含まれています。細かい利用規定については同梱のドキュメントファイルの内容に従ってください。

▼『プロジェクト・フォルダ形式』フォルダ内の作品について

6作のゲーム（一部サンプル版）が、作者の方々のご厚意によって公開直前の状態で収録されています。ダウンロードした吉里吉里を展開すると得られるtemplateフォルダと同じサブフォルダ構造を持っており、これをリリーサーでexeにまとめると公開用の実行形式ファイルになるので、リリーサーの使い

方を練習するときにも使えるでしょう。また、すべての素材やシナリオが丸見えなので、ksファイルの中身を読みながら動作を確認していくことによってKAGシナリオを書くときの参考になるかもしれませんが、画像や音響素材の利用は本書を購入された方の個人的な学習用途に対してのみ可能なので、注意してください。

すべての作品はプロジェクト・フォルダを「krkr.eXe」にドロップすれば純粋にゲーム（一部サンプル版）として遊ぶことができますが、これらの作品のほとんどは特殊な処理を行なう上でKAG講座と同様に吉里吉里プラグインを利用しているため、吉里吉里を展開したときに得られる「kr2_218r2¥kirikiri2¥plugin」内に含まれているすべてのプラグインを「krkr.eXe」と同じフォルダ内に配置する必要があります。これを行なわずに起動すると正常動作しないので、ご注意ください。

また、その他の細かい利用規定については、圧縮ファイルを解凍すると*得られるreadme.txtまたはそれに類するドキュメントをご覧ください。

＊注
それぞれの圧縮ファイルの展開について説明します。
「KAG講座」「1999xe」「ドロヌマン」「紫月」「桃花扇」「ZODIAC」の6作品はGCA自己展開書庫形式なので、ダブル・クリックして展開先を指定すれば、そこに圧縮ファイルの中身が展開されます。
「CanvasLife」「Foster」「PictualStory-twei-」はインストーラ形式なので、インストーラの指示に従って展開してください。
「showercall」「ガラスの中の少女」「学校七不思議」「最後の夏」「夢の終わり」はLZH形式の圧縮ファイルなので、LZHファイルを展開できる圧縮解凍ソフトを利用して展開してください。
プラグインの「bubble.zip」と「collage」はZIP形式の圧縮ファイルなので、ZIPファイルを展開できる圧縮解凍ソフトを利用して展開してください。

▼『実行形式フォルダ』内の作品について

「吉里吉里2/KAG3」で制作された完成版の作品が8本収録されています。一般公開用と同様の圧縮形式（一部インストーラ形式）で収録してあるので、それぞれの作品を自分のPCにインストールするか、任意の場所に展開するなどしてお楽しみください。アニメーションや画像の移動、変数処理による謎解きやマルチエンディングなど、「吉里吉里2/KAG3」を使うとどのような作品が制作できるのか、その理解の手助けになると思います。

なお、一部作品については作者の方のご厚意からksファイルのみ公開のお許しをいただきました。これらのサブフォルダにはscenarioフォルダが存在し、その中にKAG3シナリオの書かれたksファイルが圧縮状態で入っています。画像や音などの素材がないため実行はできませんが、読むだけでも勉強になるでしょう。

▼『素材』フォルダ内のデータについて

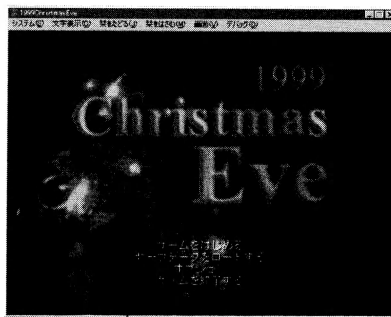
フリー素材として利用できるOggVorbis形式のBGMをご提供いただきました。この素材フォルダ内のデータだけは取り扱いが異なり、皆さんがゲームを作る上で使えそうなシーンがあったらどんどん利用してもらって構いません。もちろん、作者の方の利用規定をきちんと守ってご利用いただく必要があるので、収録されているreadme.txtをご覧ください。

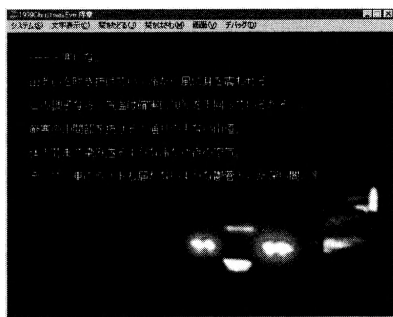
収録作品紹介

ホラー・テキストアドベンチャー『1999ChristmasEve［ver.1.999Ω］』

「横浜かまいたちファンクラブ」（YKFC）内にある「1999ChristmasEveProject」が、2000年の12月に公開したノベルタイプのテキストアドベンチャーゲームです。

ホラー系なので怖いものが苦手な方にはお勧めできませんが、「当時のKAGでどんなことができるのかをできるだけ数多く試す」という目的で、さまざまな表現形態に挑戦した、実験的な作品となっています。

現在ネット上で公開されている最新のバージョンは「1.3」で、これは吉里吉里/KAGの旧バージョンで制作されたものですが、本書には吉里吉里2/KAG3に移植した未公開の「Final β（Ω）1.999」バージョンが収録されています。バグ・フィックスは行ないましたが、一部誤動作することがあるかもしれません。予めご了承ください。なお、これは公開用完成版ではないので、再配布はご遠慮ください。

演出の見所やKAGシナリオのワンポイント

それほど大したことはしていないのですが、以下の機能を使っています。

- 別ファイルによるマクロの集中管理
- 制限時間つき選択肢
- RPG風戦闘画面の構築
- 音楽と文字表示の同期
- OggVorbisによるBGM
- 変数を利用した体力の表示とサブルーチン
- スタッフロールとエンディングリスト
- 右クリックによる未読/既読判別機能

「本格的な長編を作りたい」という理由で制作に入ったので、シナリオが長くなることは覚悟していました。このため、できるだけ読みやすい記述を心掛けましたが、いかがでしょうか。KAG使いを目指す方々のご参考になれば嬉しいです。

（文：『横浜かまいたちファンクラブ』代表　ＰＩＡ少尉）

http://www.piass.com/ykfc/

ヴィジュアル・ノベルゲーム『Canvas Life 前夜祭DISC』

家族と家庭というものは、戦後になってその関係が著しく変容しつつあります。

子供は親の所有物という父を家長とする古い家族関係の時代とは違い、最近良く耳にするのは「虐待」という言葉。子供の自我が成り立たない間、その個体が生命を維持するには親に依存しなければいけません。しかし、親が子供に依存してしまい、行き過ぎた躾や性的搾取にいたることは珍しくないです。

「子供って何だ？」「家族って何だ？」そういうところからこのゲームは生まれました。尊敬されない父親、母であることを忘れる母親、感謝することを忘れた子供………今の時代、いろいろな大切なことが豊かさに隠されておざなりになってしまっています。大切な忘れ物を届けるために、このゲームは作り出されたのです。ライトな感覚でお楽しみください。

演出の見所やKAGシナリオのワンポイント

この作品は「吉里吉里2 β 39」の時代に作られたものです。初めて吉里吉里で作った作品なので、かなり稚拙な内容になっていますが、「未読判定付きのＣＧ鑑賞モード」や「ロード画面」「音楽モード」や「スタッフルーム」など、構造的には一通りを網羅しているので、これから作ろうという方には参考になると思います。

収録するに当たって、もともとの成人指定の内容を一般向けに改変し、「ver2.18」に対応させています。ソースがめちゃくちゃ汚いのは申し訳ないです。現在までに五本の成人指定ＶＮＧをリリースしてますが、最新のは多少きれいなソースです。後は本当にベーシックな演出しかしていません。

これから死ぬまで吉里吉里でモノを作り続けていくと思います。それだけ吉里吉里には魅力があります。このツールは表現することの喜びをもたらしてくれました。みなさんにもそれが伝われば、作者冥利に尽きます。

（文：『ICONOCLASM』主宰　京秋人）

http://www.alpha-net.ne.jp/users2/ryouot/

ヴィジュアル・ノベルゲーム『シャワーコール』

『現実』という単語を辞書で調べてみましょう。そこには『頭の中で考えるだけのことではなく、現に実際こうであるという状態・事実』と書かれていると思います。

でも、こんなことを考えたことはありませんか？　人生など、誰かが見ている『長い夢』の一部分にすぎないのではないか…。

この作品の主人公は、ごく平凡な人間です。夢と現実とが交錯する中で、そんな主人公がどのように考え、行動するのか。ストーリーが進行するにつれて、微妙に変化する『現実』という単語の意味。そうしたものを、楽しんでいただければ幸いです。

演出の見所やKAGシナリオのワンポイント

演出・KAGシナリオともに、ごく平凡な構成になっていると思います。強いて見所を挙げるとすれば、土曜日・深夜のシナリオで作成した『雷鳴効果』だけかもしれません。

ほぼ一年前に作成したKAGシナリオを見てみると、何ともガサツで幼稚なタグを書いていたのだと思わず赤面。

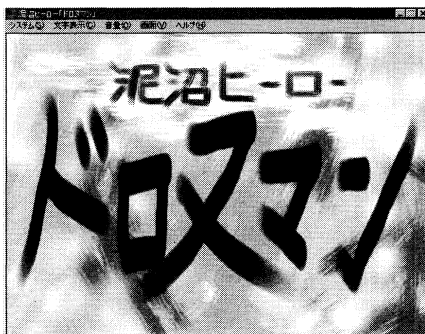
サンプルとして皆様のお役に立つかどうか甚だ疑問ではあるのですが、この程度の知識でもノベルを製作できるということをご理解いただければ本望です。

（文：『studio WALK』代表　朝から炒飯）
http://members.jcom.home.ne.jp/studio-walk/

脱力系ブラックお笑いノベル『泥沼ヒーロー　ドロヌマン』

人がいるところで「助けて〜」と叫べば、たいてい誰かが手を差し伸べて助けてくれると思います。

しかし、それで助かるとは限りません。事態が悪化するかもしれません。そればかりか、助けてほしいわけでもないのに、余計なことをしてくれるかもしれません。そんなヒーローがいたとしたら……

という感じで製作されたのが、このドロヌマンです。『泥沼ヒーロー』ですから、もちろん何かすればするほど事態が悪化します。実際にいたとしたら、絶対に来てほしくない、ハタ迷惑なヒーローの生き様を見てやってください。

演出の見所やKAGシナリオのワンポイント

本作はギャグノベル。そのため、「テンポよく」「事態が悪化する」ということを念頭に置いてネタを考えました。ある程度ネタが揃ったところで、それらを組み合わせて全体の流れを決めます。

この後に絵を描き始めるのですが、ここでKAGでどのような演出ができるのかを知っていれば、具体的にどんな絵を描けばよいかイメージしやすくなります。まずは簡単なものを作ってみることをオススメします。私も初めての練習として、ドロヌマンを作りました。

さて、全体の流れと見せ方が決まると、こんどはシステムをどうするかが決まります。プログラムに慣れている方ならば、いきなりマクロを組んで処理を効率化させることもできるとは思いますが、そうでない方はまずマクロなんて考えずにシナリオを書いてみてください。そして少し作ったところで見直してみると、似たようなことを複数記述している部分があるでしょう。その部分をマクロで統一すると、スクリプトが見やすく、書くのも簡単になります。ですから、マクロはぜひマスターすることをお勧めします（と言いつつ、私もさほどマクロ化できてるわけではないのですが。（＾＾；）

あとはやる気と時間さえあれば、きっとあなたにもゲームが作れますよ！

（文：『地球のあくび』　うにゅ〜）

http://www.earthcape.ne.jp/users/unyu/

女性向けノーマル恋愛アドベンチャー『桃花扇～Sample Version～』

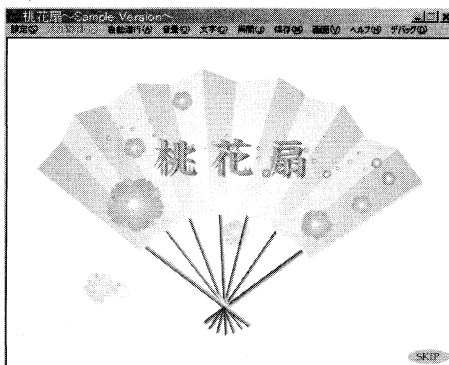
フリーソフトとして配布予定の中華風アドベンチャーの体験版プラスα。序盤の一週間がプレイできます。

プレイヤーの分身となるヒロインは、桃花（名前変更可能）おてんば娘で口の達者な女の子です。

攻略可能な男性キャラクターは全部で七人。Sample Versionには、そのうち二人が登場します。彼らとの好感度を高めていってベストエンディングを迎えるのが、ゲームの主な目的です。

体験版にシナリオを若干追加したため、半端なところで話が途切れています。続きの気になる方は、一度クリアすると出現するパスワードをホームページまでお持ちください。

演出の見所やKAGシナリオのワンポイント

いちばん苦心したのは、RPG風の戦闘シーンです（「daily¥batol.ks」。未完成）メッセージ・レイヤーをpositionタグで体力ゲージとして表示させ、敵に攻撃されるごとにゲージが減っていく仕組みになっています。

朝ごとの日付表示は「call－return」で呼ばれ、月や年が自動で進んでいきます（daily¥date.ks¥*date）。

立ち絵表示はマクロ化し、直前に表示された画像と比較して、同じならトランジション時間を短縮します（daily¥makuro_trans.ks）。

まだまだKAGの多彩な機能を使いこなせていませんが、皆様の参考になれば幸いです。

（文：『SENA Project』　グロァ）

http://fairy.vis.ne.jp/project/

ホラー・アドベンチャー『紫月』Vol.0（サンプル版）

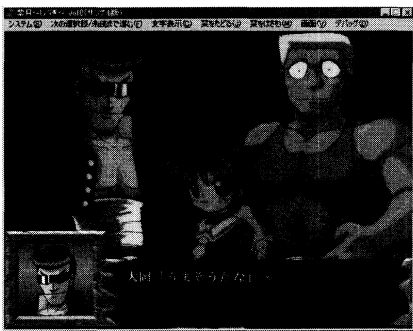
「IssisFactory」が「横浜かまいたちファンクラブ」さんと共同で開発している学園ホラー・アドベンチャーです。

完成はまだ先なのですが、PIA少尉殿のご要望に従い、序章の部分の全ソースをKAG3シナリオも含めて素材の状態で提供させていただきました。序章のみのサンプル版のため、ゲームとして遊ぶには非常に短いですが、立ち絵と顔絵を使ったアドベンチャーゲーム風の画面レイアウト構築、右クリックによる現在位置表示の他、我々が目指している効率的なマクロ定義によるシナリオ簡略化がご参考になれば嬉しいです。

分担ですが、「IssisFactory」の吉田が原画、彩色、背景CGなどのグラフィック全般を、そして「横浜かまいたちファンクラブ」さんのスタッフの方々がシナリオとサウンド、全体のコーディネイトを手掛けてくださっています。

（文：『IssisFactory』代表　絵師・吉田神一）

http://www.piass.com/issis/

演出の見所やKAGシナリオのワンポイント

アドベンチャーノベル風の画面レイアウトのため、それぞれの用途に応じた前景レイヤーを6枚定義しており、必要に応じて[laycount]でレイヤー数を調整して使っています。

また、本作はできるだけBGMの使用を抑え、効果音による臨場感と聴覚刺激による恐怖を演出する方向で制作しているので、BGM、SEともにOggVorbis形式を採用しました。特に効果音についてはW.Deeさんが制作した「LoopTuner」と呼ばれるソフトで生成できるSLIファイルを利用しています。

さらに、背景絵、顔絵、立ち絵などは、すべて「IssisFactory」の吉田神一氏にお願いし、ほとんどの立ち絵キャラクターに目パチアニメーションを採用、さらに重要なイベント絵については市販のゲーム同様1枚絵を書き下ろしてもらっています。

「IssisFactory」のポップなキャラと、YKFCのシリアスなシナリオがマッチするように気をつけながら、鋭意制作中です。

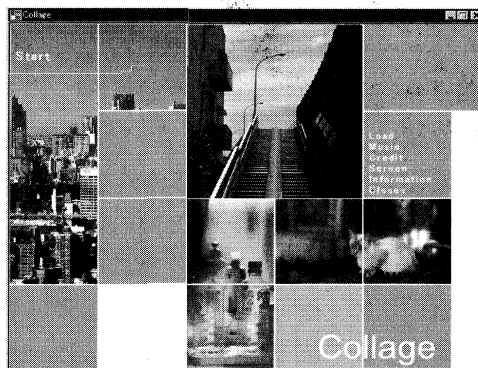
（文：『横浜かまいたちファンクラブ』代表　ＰＩＡ少尉）

http://www.piass.com/ykfc/

スタイリッシュ・ヒューマンドラマノベル『Collage』

物語というのは一人の人生の一部分を垣間見るという行為だと思いますが、脇役にも同様に人生があり物語があります。その物語においてはその脇役が主人公であり同様に脇役が存在し、その脇役にもまた人生があり...

そう考えると一人の人間の人間関係というのはきわめて複雑です。その相関関係や感情の行き来を完全に認識することは不可能でしょう。

「Collage」の制作は、そのような謎を少しでも解き明かしたいというのが動機のひとつとしてあります。もちろんそんなことは不可能ですが......まあ、あまり難しいことは考えずに、「こいつとこいつがこうなるのかよ！」みたいな感じで楽しんでいただければ幸いです。

演出の見所やKAGシナリオのワンポイント

この作品はmoveタグを多用することによって動きのある画面を構成し、プレイヤーを飽きさせないように工夫したつもりです。また音楽や効果音と画像のシンクロも意識してたので、そちらも楽しんでいただけるのではないかと思っています。

吉里吉里２は非常に高機能でアドベンチャーゲームを製作するということで想像されるイメージは、ほぼすべて再現することができます。はじめは少々面倒に感じるかもしれませんが、使えば使うほどできることの多さに開発が楽しくなると思います。とにかくやってみてください！

（文：『Homegorosi PROJECT』　コミネト）

http://www.geocities.co.jp/Bookend-Ryunosuke/3968/

コマンド選択式アドベンチャー『Foster』

温泉宿での殺人事件の謎を追う、オーソドックスなコマンド選択式アドベンチャーゲームです。

殺人事件が発生した温泉宿で、事件の真相を追うために主人公が宿での聞き込みを開始します。

ゲーム中では時間が流れており、無駄な捜査をすると事件の真相にたどり着くことはできません。

イラストはネット上でも活躍されている桜木晶さんにご協力いただきました。どれもすばらしく、これだけでも一見の価値はあると思います。

（文：『Red Letter』　東 真哉）

http://hgashi.tripod.com/redletter/

恋愛シミュレーション『PicturalStory -twei-』

「PicturalStory」の続編で、画家を目指す青年が女の子たちと繰り広げる恋愛(?)物語がテーマの恋愛シミュレーションゲームです。

前作の設定は、修行のためにある街にやってきた絵描きを目指す青年が芸術の発展に力を入れているその街で、画家としての腕を磨きつつ、絵のモデルを探し、恋をする――そんな物語でしたが、この続編では絵のことなど何も知らない青年が前作の舞台である街を訪れ、いろいろな物語を繰り広げます。ゲームの目的は前作同様3年の間に女の子と親しくなって、絵のモデルになってもらうことです。

（文：『Red Letter』　東 真哉）

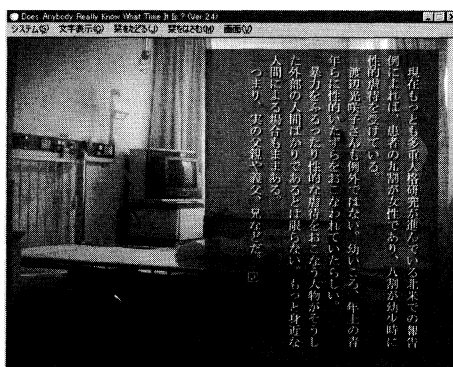
http://hgashi.tripod.com/redletter/

サイコホラーノベル『ガラスの中の少女』

11の人格を持つ解離性同一性障害の患者の、妄想の中の殺人事件。しかし現実と妄想とが徐々に交錯してゆき、主人公の周辺で異常な事態が起こり始める……。

「ガラスの中の少女」は、多重人格をテーマに描くサイコホラーノベルです。各章の最後でキーワードを選択・取得し、クライマックス手前でキーワードを組み合わせて推理をするという、やや特殊なシステムを採用しています。問題編と解決編にわかれた犯人あて推理小説の、ホラー版といったところです。

「最後の夏」と同様、「パソコンで読む小説」を意識して作ったものです。背景の多くは合成写真ですが、自己主張があまり激しくならないように、かつ絵になるように調整しました。音楽も、しっとりと落ち着いていて美しいものをお借りしました。トータルバランスには気を配ったつもりです。

演出の見所やKAGシナリオのワンポイント

デモシーンなどで音楽と画面のタイミングをあわせるためには、[resetwait]、[wait mode="until"]を使います。

ただし、画像を読み込むためのタイムラグが生じるので、「touchImages」を使って画像をキャッシュに入れておくか、「assignImages」を使って順次処理をする必要があります。

また、作業フォルダで吉里吉里を走らせるのと、アーカイブ化した状態とでは、タイミングが変わってきます。

わたしはこのあたりを（実はこれに限らず全編を）行き当たりばったりで適当に作業していました。何回調整したか数え切れません。製作には計画性が必要だと痛感しました。

（文：『林檎坂通信』管理人　SUZUNE）

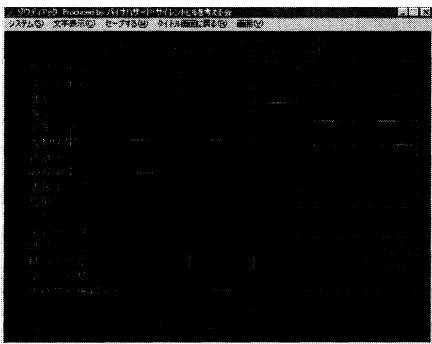
http://www2.odn.ne.jp/suzune/

ホラー・アドベンチャーノベル『ZODIAC』

「ホラー・アドベンチャー」と「ノベル」。このまったく異なる2つのジャンルを融合させた、新しい感覚のゲームです。

雪に閉ざされた秘境古代都市、ゾウディアックにやってきた主人公は、突然の雪崩に巻き込まれて仲間と離れ離れに。仲間を捜しながら助けを呼ぶべく歩き回る主人公ですが、彼の前に見たこともない怪物や謎めいた現象が立ちはだかります。ゾウディアックで何が起こったのか。そして主人公は無事に仲間を見付けて帰還できるのか。先の読めないストーリーがあなたを待っています。

また、ホラー・アドベンチャーゲームには欠かせない謎解きにも力を入れています。プレイヤーはステージ中に散りばめられたヒントを集めながら、謎を解いて先に進んでいきます。謎解きの難易度は高めなので、謎解きが得意な方、市販のソフトの謎解きが簡単すぎるという方にお薦めします。その他にも敵との戦闘、選択肢によって変化するストーリー、マルチエンディングなど、プレイヤーを飽きさせないように工夫をしています。ぜひ一度プレイして、この楽しさを味わってください。

演出の見所やKAGシナリオのワンポイント

本作品では複数の人間がスクリプトを担当したので、演出の統一に注意を払いました。

また、BGMや背景画像のフェードアウト/インも、あまりやりすぎるとユーザーにストレスを与えてしまうし、まったく意識しないと呆気のないゲームになってしまいます。そのバランスは製作者によって違ってくるのでしょうが、演出上非常に重要な部分だと思います。

本作品では行末クリック待ち（[l]）の代わりにウェイトタグを使用していますが、ユーザーの評判はイマイチでした。製作者にとって重要なのは、自分が何を作りたいのかと、ユーザーが何を望んでいるのかを常に考えながら製作することかもしれません。

（文：『ゾウディアック』制作指揮　陽炎）

http://www.zodiac-series.com/

学園ホラー・アドベンチャー『学校七不思議』

学生なら、誰もが必ず一度は耳にする怪談話。七つあると言われるけれど、七つ目はなぜか誰も知らない。「なぜ誰も知らないのに七不思議？」「七つ目を知ってしまうとどうなるのか？」。単に学校の怪談話を並べるだけでなく、その謎について考えた物語をゲームにしてみました。

忽然と消えた二人の女生徒。彼女たちが消えたのも、こんな暑い夏の日だった…。現在学生の人はもちろん、かつて学生だった人も、母校を思い浮かべながらプレイしてみてください。二部構成になっているので、ボリュームはあるかと。マルチエンディングのため、すべての結末を見るのは大変ですが、クリアすると“おまけ”がありますので、お楽しみに。

演出の見所やKAGシナリオのワンポイント

途中の選択肢により、物語はさまざまに分岐しますが、共通イベントはサブルーチンを使っています。そのぶん、シナリオは分かりやすくまとめることができました。

（文：『銀の盾』　藍澤風樹）

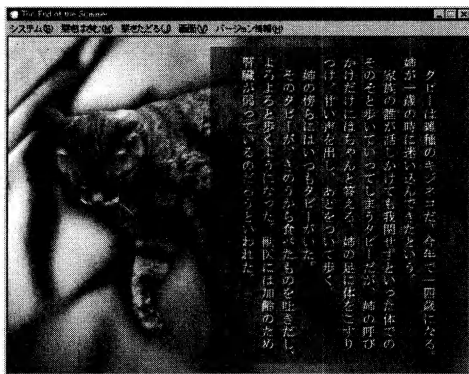
http://hp.vector.co.jp/authors/VA022686/

ファンタジック・ショートショートノベル『最後の夏』

「最後の夏」は、失踪した猫をめぐって繰り広げられるファンタジックなショートショートで、ゲーム要素はいっさいありません。

「パソコンで読む小説」を意識して作りました。縦書きの明朝体を採用していたり、右半分のみに文字表示ウインドウをかぶせているのも、ウインドウを本に見立てたからです。左側はさし絵の部分というわけです。

サウンドノベルと呼ばれるゲームでは、インタラクティブ感を高めるために、「主人公が見ている映像」または「主人公をとらえたカメラの映像」を表示する場合がほとんどだと思います。挿絵方式は、独特といえばいえるかもしれません。続く「ガラスの中の少女」では、さらにイメージ映像に近いものが多用されることになります。

演出の見所やKAGシナリオのワンポイント

「吉里吉里1/KAG 2」のスクリプトだということで、今回シナリオ・ファイルはCDへの収録を見送らせていただきました。

ほとんどが基本機能のみで、テクニック的にもあまり凝ったことはしていません。強いていえば冒頭の、文字が浮かび上がってくるところくらいでしょうか。[current layer="message1" withback=true]という状態で[delay speed="nowait"]に設定して文章を記述、このレイヤのトランジション処理を行います。

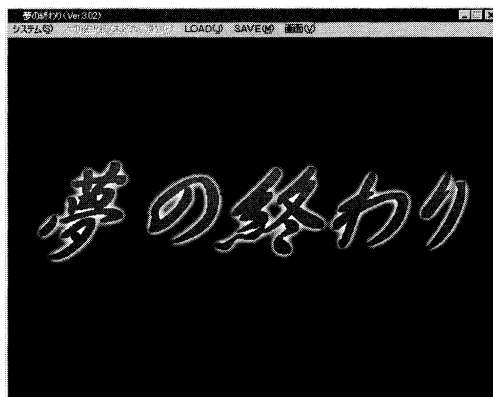
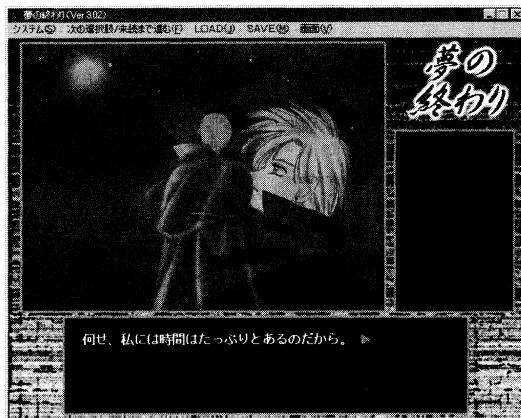
次に、[ct]でメッセージ・レイヤー0にフォーカスを戻し、先ほどと同じ文章を記述した後、[layopt layer="message1" page="fore" visible="false"]にしています。これを繰り返して、1行ずつ増やした文章を順次表示していっているわけです。

(文：『林檎坂通信』管理人　SUZUNE)

http://www2.odn.ne.jp/suzune/

コマンド選択式アドベンチャー『夢の終わり』

アドベンチャーゲームと言えば、昔は「見る」「調べる」などのコマンドを選んで進めていく形式がほとんどでしたが、最近はあまり見掛けません。それなら、自分で作ってしまおうということで、昔懐かしい形式で制作しました。

永遠の刻を生きる、ラルクという名の美しい吸血鬼がいる。彼は退屈と停滞を嫌い、旅を続けていた。不老不死の彼を今宵待ち受けるは、死の臭いに満ちた村と、朽ちた教会で祈り続けるシスター。夢を見ていた者は誰？　その望みは…？

演出の見所やKAGシナリオのワンポイント

コマンドの表示はマクロでメッセージ・レイヤーの雛形を作っておき、通常メッセージ表示時と切り替えて使っています。

これに限らず、同じ形式を多用するときはマクロを活用するととても便利ですよ。フレーム内画像の表示位置も、マクロで設定しました。

（文：『銀の盾』　藍澤風樹）

http://hp.vector.co.jp/authors/VA022686/

「吉里吉里/KAG推進委員会」について

「吉里吉里/KAG推進委員会」(以下、「委員会」)は、吉里吉里/KAGユーザーが集うインターネット上のコミュニティで、吉里吉里/KAGの開発者であるW.Deeさんが正式に認可し、ほぼ常駐してくださっています。ですが、開発者のW.Deeさんにはできるだけ吉里吉里/KAGの開発に専念してもらいたいという理由から、サイトの管理は私（ＰＩＡ少尉）が不肖ながら務めさせていただいています。

現在の委員会サイトはユーザーの皆さんが質問や意見を書き込める掲示板や、作った作品を告知するための自由登録リストなど、「参加型」のコンテンツが中心になっています。

このように当委員会は、開発者であるW.Deeさんに認可いただいた唯一の支援系公式サイトとして、吉里吉里/KAG本体や、吉里吉里/KAGによって制作された作品に関する情報の一元化を目指しています。

現在のところ、当委員会は以下の4つの目的のために存在しています。

①吉里吉里/KAGに関する各種情報の告知場所として
②同ツールを使う制作者のスキルアップの場として
③エンドユーザーと開発者の相互連絡の場として
④エンドユーザー同士の情報交換の場として

吉里吉里/KAGを使って何かを作ろうと考えている方は、ぜひ委員会へご参加ください。ネット上の最低限のマナーが守れる方ならどなたでもご参加いただけますので、吉里吉里/KAGに興味があるけど何が何だか分からない方から、吉里吉里/KAGで制作した作品を告知したい方まで、自由にご利用ください。

なお、当委員会には参加のための面倒な手続きや審査などは一切ありませんし、それよりも何よりも、入会や退会などの概念すら存在しません。ご自分が「ぼくは推進委員だ！」と自己表明した瞬間から、あなたは推進委員です。

詳細については下記のサイトをご覧ください。

吉里吉里/KAG推進委員会
http://www.piass.com/kpc/

演算子一覧

分類	演算子	使い方	意味
関数呼び出しなど	( )	(a)	演算の優先順位の変更
	( )	a( )	関数 a を呼び出す
	[ ]	a[b]	オブジェクト a のメンバ b にアクセス(間接メンバ選択)
	.	a.b	オブジェクト a のメンバ b にアクセス(直接メンバ指定)
	後置 ++	a++	a に 1 を足す (式全体としては演算前の値を示す)
	後置 --	a--	a から 1 を引く (式全体としては演算前の値を示す)
	後置 !	a!	a の表わす文字列を式として評価
	incontextof	a incontextof b	a のコンテキストオブジェクトを b に変えたもの
	int	int a	a を整数型に変換
	real	real a	a を実数型に変換
	string	string a	a を文字列型に変換
単項演算子など	前置 !	!a	a の真偽を否定
	~	~a	a のビットを反転
	前置 ++	++a	a に 1 を足す (式全体としては演算後の値を示す)
	前置 --	--a	a から 1 を引く (式全体としては演算後の値を示す)
	new	new a( )	クラス a のオブジェクトを作成する
	invalidate	invalidate a	オブジェクトaを無効化する
	isvalid	isvalid a または a isvalid	オブジェクト a が有効ならば “真”

単項演算子など	delete	delete a	変数またはメンバ a を削除
	typeof	typeof a	a の型を調べる
	#	#a	a の文字コード
	$	$a	a のコードが表す文字
	単項 +	+a	a を数値型にする
	単項 -	-a	a の正負を逆にする
	単項 *	*a	プロパティオブジェクト a にアクセス
	instanceof	a instanceof b	a がクラス b のオブジェクトならば“真”
乗除余算演算子	%	a % b	a を b で割ったあまり
	/	a / b	a を b で割る (値は実数として扱われる)
	¥	a ¥ b	a を b で割る (値は整数として扱われる)
	*	a * b	a と b をかける
加減算演算子	+	a + b	a と b を加算 (文字列に対しては結合)
	-	a - b	a から b を減算
ビットシフト演算子	>>	a >> b	a を b 回右に算術シフト (符号ビットを保つ)
	<<	a << b	a が b 回左にシフト
	>>>	a >>> b	a を b 回右に論理シフト (上位ビットを 0 で埋める)

<table>
<tr><td rowspan="4">比較演算子</td><td><</td><td>a < b</td><td>a が b より小さければ“真”</td></tr>
<tr><td>></td><td>a > b</td><td>a が b より大きければ“真”</td></tr>
<tr><td><=</td><td>a <= b</td><td>a が b と同じか、あるいは
a が b より小さければ“真”</td></tr>
<tr><td>>=</td><td>a >= b</td><td>a が b と同じか、あるいは
a が b より大きければ“真”</td></tr>
<tr><td rowspan="4">同定演算子</td><td>==</td><td>a == b</td><td>a と b の値が同じならば“真”</td></tr>
<tr><td>!=</td><td>a != b</td><td>a と b の値が異なれば“真”</td></tr>
<tr><td>===</td><td>a === b</td><td>a とbの型と値が同じならば“真”</td></tr>
<tr><td>!==</td><td>a !== b</td><td>a と b の型が異なるか、あるいは値が異なれば“真”</td></tr>
<tr><td>ビット AND 演算子</td><td>&</td><td>a & b</td><td>a と b のビットごとの AND (論理積)</td></tr>
<tr><td>ビット XOR 演算子</td><td>^</td><td>a ^ b</td><td>a と b のビットごとの XOR (排他的論理和)</td></tr>
<tr><td>ビット OR 演算子</td><td>|</td><td>a | b</td><td>a と b のビットごとの OR (論理和)</td></tr>
<tr><td>論理 AND 演算子</td><td>&&</td><td>a && b</td><td>a が“真”かつ b が“真”ならば“真”</td></tr>
<tr><td>論理 OR 演算子</td><td>||</td><td>a || b</td><td>a が“真”または b が“真”ならば“真”</td></tr>
</table>

条件演算子	? :	a ? b : c	a が“真”ならば b を評価、そうでなければ c を評価
代入演算子	=	a = b	a に b の値を代入
	<->	a <-> b	a と b の値を入れ替える
	&=	a &= b	a = a & b
	|=	a|= b	a = a b
	^=	a ^= b	a = a ^ b
	-=	a -= b	a = a - b
	+=	a += b	a = a + b
	%=	a %= b	a = a % b
	/=	a /= b	a = a / b
	¥=	a ¥= b	a = a ¥ b
	*=	a *= b	a = a * b
	&&=	a &&= b	a = a && b
	||=	a|| = b	a = a|| b
	>>=	a >>= b	a = a >> b
	<<=	a <<= b	a = a << b
	>>>=	a >>>= b	a = a >>> b
並列演算子	,	a , b	a を評価したあとに b を評価
if演算子	if	a if b	b が真ならば a を評価

優先順位は、上にあげたものほど高くなりますが、同じ分類の演算子間の優先順位は同じです。

索引 index

五十音順

アルファベット

記号/数字

タグ

属性

あとがき

「KAG」と「TJS」、いかがでしたでしょうか。

この本ではKAGとTJSの基本を学ぶことができたと思います。「基本的」というのはどういうことかというと、KAGもTJSも応用範囲が広く、考えつく限りにしても、応用的なことをすべて紹介するとなると、膨大な量になってしまいます。しかしながら、今回執筆させて頂いた内容は、吉里吉里、そしてKAGとTJSのエッセンスを伝えるには充分であると思います。

知れば知るほどに応用は広がりますが、同時にさらに知りたい部分も増えてくるかと思います。この本を読まれた方々は、吉里吉里/KAGの世界をもっと知りたいと思うようになると思います。この本に書かれていない多くの情報は、吉里吉里/KAGのSDK(ソフトウェア開発キット)付属のドキュメントに書かれています。インターネット上の検索エンジンも役に立つでしょう。

また、今回CD-ROMに収録させていただいた作品の中には、そのソースを閲覧できるものがあります。他の人の作品をよく研究することも、より深い理解につながる良い方法だと思います。

「吉里吉里/KAG推進委員会」もぜひご利用ください。インターネット上にある講座の自動登録リストがあります。掲示板にお越しいただいて、質問すれば、きっと答えが返ってくると思います。

これらの環境は、吉里吉里/KAGをサポートする多くの方々によってつくられたものです。「謝辞」にも書いた通りですが、再度感謝申し上げたいと思います。

吉里吉里/KAGはソフトウェアのソースが公開されています。ソースを公開することにより、ほかの開発者の方々のご意見を広く募ることができたり、あるいは実際に開発に参加していただくということが可能になります。また、ソース公開を前提とするプログラミングではいい加減なことはかけません。必然的にプログラムをきれいに書こうという意識を、自分を含めた開発者自身にもたせることができ、結果的により良いソフトウェア作りにつながっていくのだと思います。

吉里吉里は発展途上のソフトウェアです。日々改良、機能追加が行なわれています。これは作者の力の続く限り続けていきたいものです。どうぞ、皆様の暖かいご支援を引き続き賜りたく、お願い申し上げます。

2003年5月吉日 W.Dee

思い起こせば1999年の12月。

軽い気持ちでダウンロードした吉里吉里のサンプルに衝撃を受け、こんなすごいツールを制作しているのはどんな人なのだろうとサイトを訪問させていただいたところ、おや、と思える一文を発見しました。

1999/8/24 進捗
なんか、吉里吉里本体は開発が止まっているというか、なんというか。使ってくださっている人からのレスポンスが今のところ皆無に等しいので(泣)

モニタに向かって「こんなすごいソフトなのになぜだ!?」と声に出して叫んだ自分は、「このままではいかん」と焦りました。4ヶ月も前の発言でこの状態では、すでに開発者のモチベーションがどん底まで落ちており、ひっそりと闇に葬られてしまっているかもしれぬ。メールを出そう。出して、ここに一人でも応援している人間がいることを伝えねば！
——と、こんな経緯で衝動的にW.Deeさんに初メールを差し上げたそのときから、私の「KAGと心中」人生は始まったのであります。

本書にも書かせてもらいましたが、このソフトは「これをしたい」という要求のほとんどに応じてくれます。最初に敷居をまたぐだけの努力さえすれば、あとはどんどん手に馴染んできます。アドベンチャーやノベルだけではなく、シミュレーションゲームやパズルなどはもちろん、プレゼンテーション・スライドや学習教材などにも幅広く利用できます。私自身、それほど使いこなしているわけではないのがお恥ずかしいのですが、このソフトを学習される方のために必死になって書いた本なので、本書の内容が少しでも皆様のお役に立てたら嬉しいです。

最後に、この場をお借りしてこの本を世に出すためにご協力いただいた皆様への謝辞を述べさせていただきます。共著に私のような1ユーザーを抜擢して下さった吉里吉里/KAG開発者のW.Dee様、本稿内の立ち絵や背景CGのみならずソースレベルの作品を快くご提供くださったIssisFactoryの吉田神一様と『地球のあくび』のうにゅ～様、ソースレベルでの収録をご快諾下さった京秋人様、朝から炒飯様、グロァ様、作品収録のご協力をいただいた藍澤風樹様、SUZUNE様、東様、陽炎様、コミネト様、素材を提供してくださったDauge様、お世話になった吉里吉里/KAG推進委員会の皆様、毎晩遅くまで原稿を書く自分の心身を支えてくれた我が妻、そしてこの本を手に取って下さったあなた。
本当にありがとうございました。

2003年5月吉日　PIA少尉

[著者略歴]

W.Dee

本業はソフトウェア開発。C/C++やPerl、PHPを操る。

趣味はフリーソフト作り。吉里吉里もそんなフリーソフトの一つで、最初のバージョンを公開してから4年。もともとは自分たちでなんか一本ゲームでも作ろうとしていて、じゃあなにかここで永く使える汎用的なものを作っておこうと思ったら、システム・プログラムだけが一人歩きして現在にいたる。

●kikyou.info

http://kikyou.info/

ＰＩＡ少尉

大学時代、ASCII-NET、PC-VAN、NIFTY-Serveのパソコン通信にはまり、デビューして以来このハンドルでネット活動を続け、今年でネット歴15年。

大学卒業後、勢いだけで大手ソフトハウスに入社するも6年間の勤務に疲れ、休職して海外に逃亡。

20世紀末、ノストラダムスの予言を母国で迎えるために帰国。その後平凡なサラリーマンに戻り、結婚して家庭をもつという日本の典型的なオヤジ道を歩くことに。

今ではKAGと花を愛しつつ悠々自適な日々を送る好々爺。趣味はガーデニングとKAGによるゲーム制作。好きな樹木はメガネヤナギとムクゲ。

●横浜かまいたちファンクラブ

http://www.piass.com/ykfc/

《質問に関して》

本書の内容に関するご質問は、

①返信用の切手を同封した手紙

②往復はがき

③FAX(03)5269-6031

（ご自宅のFAX番号を明記してください）

④E-mail　iomook@kohgakusha.co.jp

のいずれかで、工学社第二I/O編集部宛にお願いします。

電話によるお問い合わせはご遠慮ください。

I/O BOOKS　タグでノベルゲームが簡単にできる！

吉里吉里/KAGではじめるゲーム制作

平成15年5月20日　初版発行　©2003

著　者　W.Dee/PIA少尉

編　集　第二I／O編集部

発行人　星　正明

発行所　株式会社工学社

〒160-0003 東京都新宿区本塩町23番地 第2田中ビル7F

電　話　(03)5269-2041(代)［営業］

(03)5269-6041(代)［編集］

振替口座00150-6-22510

※定価はカバーに表示してあります。

印刷：（株）シナノ　　ISBN4-87593-426-2